Black Saint InDex

P3rL Files
P3rL RegExp
The Last Spell

How To Open File

لغة البيرل من اللغات البرمجية التي تدعم التحكم بالملفات البرمجية وهناك دالة من الدوال ال Built-in في لغة البيرل والدالة التي تقوم بهذه العملية هي دالة *open*
ويكون التمثيل البرمجي العام لهذه الدالة كألاتي من خلال هذا الكود

Code (1)

```
open (FH,path);
```

هذا الاسلوب البرمجي العام لتمثيل دالة ال *open*حيث يتم استخدام الدالة ومن ثم يتم استخدام ال *File Handle* وبعدها يتم وضع المسار الخاص بالملف اما عن ال *FH* فلا يشترط ان يتم استعمال هذين الحرفين دائما اي حرف اي حرف يود المبرمج ان يستخدمها فهذا ممكن شريطة ان تكون في حالة بعيدا عن التمثيل العام يكون التمثيل *UC* الاسمي او الواقعي لملف موجود فعليا على النظام تتم كما يلي

Code (2)

```
open (FH,"/home/spawn/aa.pl");
```

تم شرح الاسلوب البرمجي والاسلوب الفعلي لتمثيل دالة ال *open* في كل من الكود *1* و الكود *2*

How To Print

اما اذا كان الملف البرمجي الذي يتم التعامل معه هو ملف يحتوي على *Content* بغض النظر اذا كان المحتوى الخاص بهذا الملف هو كود برمجي او مجرد احرف فأن الاسلوب الاستدعاء في كل من الحالتين هو الاستدعاء ذاته

**Code (3)*

```
open (FH,"/home/spawn/aa.pl");
while (<FH>){
print $_,"\n";
}
```

هذه هي الطريقة التي يتم اتباعها في لغة البيرل لطباعة محتوى ملف ما موجود على النظام وان ناتج تنفيذ الكود كألاتي في الشكل أدناه

```
open (FH,"/home/spawn/aa.pl");
while (<FH>){
print $_,"\n";
}
1 2 3 4 5 6 7 8 9 10   OutPut
```

Figure(1)
Print The Content

How To Write

من العلميات التي تتم على برمجة الملفات هي عملية الكتابة على الملفات و هذه العملية تتم ايضا من خلال الدالة ذاتها ولكن مع بعض الاختلافات وتتم عملية الكتابة كما يلي

الطريقة الاولى من عملية الكاتبة على الملفات ويكون التمثيل البرمجي لهذه العملية كما يلي

*Code (4)

```
open (FH,">/home/spawn/aa.pl");
print FH "We are perl programmers";
```

واذا تم تنفيذ هذا المقطع البرمجي فأن الناتج من عملية التنفيذ تكون كألاتي

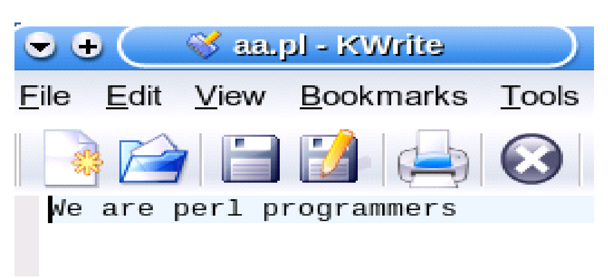


Figure (2)
Writing To File

الطريقة الثانية لعملية الكتابة على الملفات تتم كما يلي من خلال هذا المقطع البرمجي

*Code (5)

```
open (FH,">>/home/spawn/aa.pl");
print FH "We are good programmers";
```

للوهلة الاولى يبدو ان كل من الطريقة الاولى والطريقة الثانية هي طرق متشابهة ولكن اذا تم التدقيق في ال Figure(3) سوف يتم ملاحظة الفرق الاساسي بين هاتين الطريقتين

```
open (FH,">>/home/spawn/aa.pl");
print FH "We are good programmers";
open (FH,">/home/spawn/aa.pl");
print FH "We are perl programmers";
```

Figure(3)
Main Difference

اولا

*Code(6)

```
>>
```

هذا الاسلوب من الكتابة على الملفات هو اسلوب يعتمد على الاضافة الجديدة مع الابقاء على المحتوى السابق والمقطع البرمجي ألاتي سوف يوضح الفرق بين الطريقتين بصورة موسعة اكثر

*Code(7)

```
open (FH,">>/home/spawn/aa.pl");
print FH "9 8 7 6 5 4 3 2 1";
```

يوضح ال*Figure(4)* شكل الملف قبل ان يتم تطبيق المقطع البرمجي أعلاه

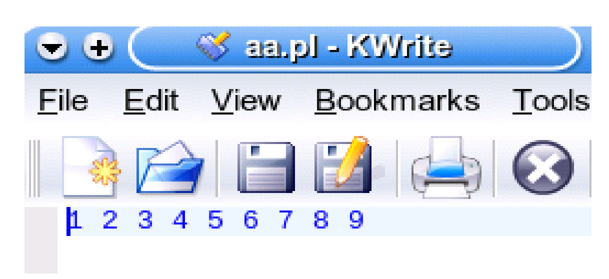
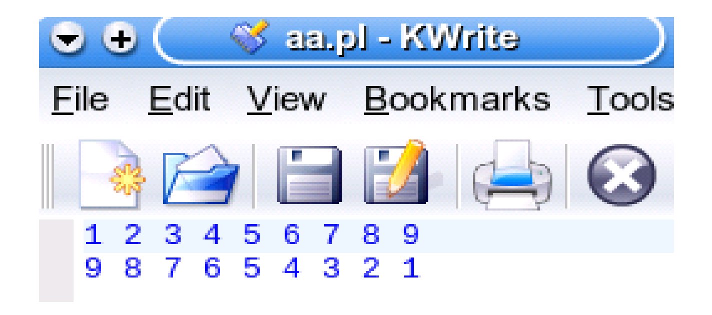


Figure (4)
B4 Execution

حيث يوضح الشكل أعلاه ان الملف الذي يحمل الاسم *aa.pl* هو عبارة عن ملف يحتوي على سطر واحد وهذا السطر هو عبارة عن ارقام من *(1-9)* فقط

اما اذا تم تنفيذ المقطع البرمجي *(7)* فأن ناتج تنفيذ هذا المقطع موضح بال*Figure(5)*



Figure(5)
After Execution

يوضح الشكل أعلاه ان اذا تم استعمال طريقة السهمين سوف يتم الاحتفاظ بالنص القديم ويتم اضافة النص الجديد عليه اي بدون خسارة البيانات القديمة

ثانيا

Code(8)

```
>
```

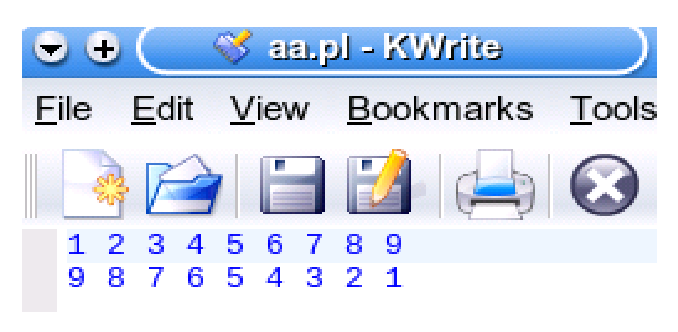
هذا الاسلوب من الكتابة على الملفات يعتمد على اسلوب اضافة النص الجديد مع حذف النص القديم وعدم الاحتفاظ به

والمقطع البرمجي ألاتي سوف يوضح هذا الاسلوب بصورة موسعة اكثر

Code(9)

```
open (FH,">/home/spawn/aa.pl");
print FH "10 20 30 40 50 60 70 80 90";
```

ويوضح ال*Figure(6)* شكل الملف ومحتواه قبل تنفيذ المقطع البرمجي أعلاه



Figure(6)
B4 Execution

يوضح ال*Figure(6)* ان الملف الذي يحمل الاسم *aa.pl* هو عبارة عن ملف يحتوي على سطرين يحتوي على تسلسل عددي من ال*(1-9) (9-1)* فقط

اما اذا تنفيذ المقطع البرمجي *(9)* فأن ناتج تنفيذ هذا المقطع البرمجي هي كالأتي

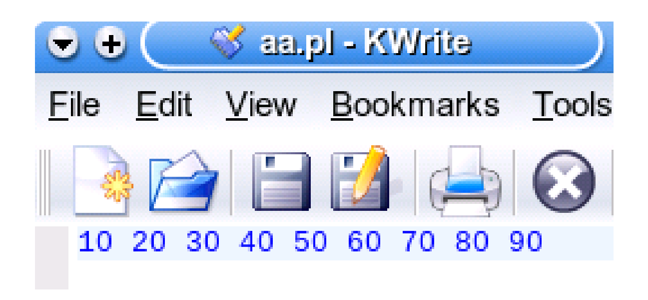
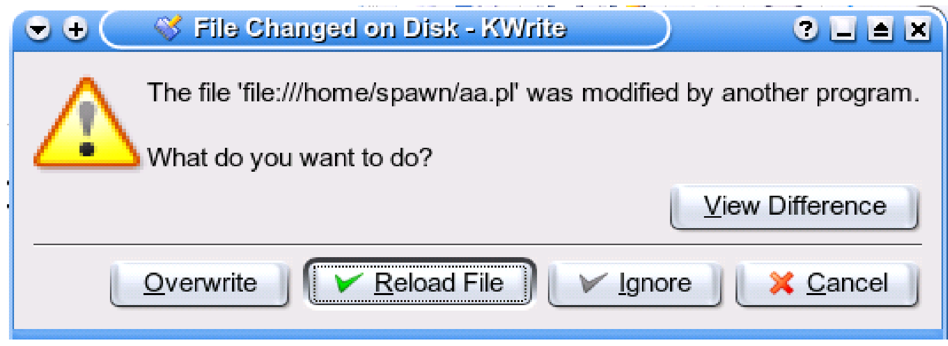


Figure (7)
After Execution

يوضح الشكل *(7)* ناتج تنفيذ البرنامج مع ملاحظة انه تم الغاء كافة محتويات الملف وأعتماد التغييرات الجديدة على الملف

*Note

تجدر الاشارة الى نقطة هامة وهي انه اذا تم تنفيذ اي عملية من هاتين العمليتين التي سبق ذكرهم على اي ملف وكان الملف الذي تتم الكتابة عليه ملف مفتوح فسوف تظهر رسالة تحذيرية للمستخدم تكون كالأتي كما هو موضح بال*Figure(8)*



Figure(8)
Warning Msg

How To Read File

احدى الطرق التي توفرها لغة البيرل لقراءة الملفات هي باستخدام دالة القراءة

*Code(10)

```
read
```

وتعتبر هذه الدالة من الدوال المبنية في لغة البيرل او ما يطلق عليها اسم *Built-in* ويكون التمثيل البرمجي لهذه الدالة كما يلي من خلال هذا المقطع البرمجي

*Code(11)

```
read (File Handle,$var,offset);
```

هذا الاسلوب البرمجي العام لتمثيل هذه الدالة اما عن طريقة استخدام هذه الدالة في برناومج فعلي لكي يتم تنفيذها على ملف موجود فعلا على النظام فهذه الطريقة يكون تمثيلها البرمجي كما يلي من خلال هذا المقطع البرمجي

*Code(12)

```
open (FIL,"/etc/services");
        read (FIL,$a,100);
print $a;
```

واذا تم تنفيذ المقطع البرمجي أعلاه فان الناتج من عملية التنفيذ هذه تكون كما يلي من خلال ال *Figure(9)*

```
# /etc/services:
# $Id: services 105871 2005-12-07 06:53:01Z flepied $
#
# Network services, Interne
```

Figure (9)
OutPut Of Read

عن هذا المقطع البرمجي تم استعمال دالة ال *open* بالصورة العادية التي يتم استخدامها بها ومن ثم في الخطوة التالية تم استعمال دالة القراءة في الخطوة الخاصة بدالة القراءة يجب ان يتم ملاحظة ما يلي

اولا

ان ال *FileHandle* الذي يتم استخدامه في دالة ال *open* يجب وبشكل ملزم ان يكون نفس ال *FileHandle* الذي يتم استعماله في دالة القراءة يعني بكلمات اخرى اكثر توضيحا المقصود هو

```
open (FIL,"/etc/services");
        read (FIL,$a,100);
print $a;
```

Figure (10)
FileHandle

يوضح الFigure(10) ال FileHandle التي تم استخدامها في المقطع البرمجي(12)
ومن ثم تم استعمال المتغير يحمل الاسم a$الذي يعمل على خزن ال offsetالذي ستتم قراءتها من الملف اي أن بعدد الارقام التي سيتم وضعها في داخل المقطع البرمجي سوف يتم قراءة احرف من الملف الذي وضع في دالة ال open في المقطع البرمجي (12) تم ادخال الرقم (100) اي في هذه الحالة سوف يقوم البرنامج في هذه الحالة بقراءة (100) حرف من الملف المطلوب قراءته وهكذا العملية تستمر اي أن الرقم وعدد الاحرف في حالة ترابط مع بعضهم اي ان عدد الارقام الموجودة في ال offset يساوي عدد الاحرف التي سوف يتم قراءتها من الملف

How To Assign

من الممكن ان يقوم المبرمج بعملية اسناد لمسار الملف الذي يعمل عليه الى متغير من نوع scalarوهذا النوع من العمليات يتم تمثيله كما يلي من خلال هذا المقطع البرمجي ألاتي

*Code(13)

```
$path="/home/spawn/aa.pl";
open (FH,$path);
while (<FH>){
print $_,"\n";
}
```

المقطع البرمجي أعلاه هو عبارة عن برنامج بسيط فقط تم اسناد مسار الملف فيه الى متغير وتم التعامل مع هذا المتغير على انه مسار الملف كما هو موضح في المقطع البرمجي (13) وناتج تنفيذ هذا المقطع البرمجي هو كالأتي

```
10 20 30 40 50 60 70 80 90
```

Figure(11)
Assign The Path

How To Report Errors

في بعض الحالات التي يتم كتابة ملف يتعلق ببرمجة الملفات في لغة البيرل فهذا يعني ان المقطع البرمجي لا بد له ان يتضمن مسار لملف ما فماذا يحدث عند هذه الحالة لغة البيرل قد وفرت دالة برمجية من نوع ال Built-in تتكفل بهذه العملية و هذه الدالة هي دالة

*Code(15)

```
die
```

اما عن طريقة استعمال هذه الدالة في مقطع برمجي فعلي فأن العملية تتم بالشكل ألاتي

*Code(16)

```
open (FH,"/home/spawn/aas.pl")
|| die "Cannot find  such file";
```

وان ناتج تنفيذ هذا المقطع البرمجي يكون كالأتي بالشكل ألاتي

```
open (FH,"/home/spawn/aas.pl") || die "Cannot find  such file";
Cannot find  such file at - line 1.
```

Figure(12)
Die Func

How To Count Lines

ان برنامج ال *Count Lines* من البرامج الشائعة في اغلب اللغات البرمجية وهذا البرنامج من الممكن ان تتم برمجته في لغة البيرل بأسلوب سهل ومرن جدا ويكون كألاتي

*Code(17)

```
open (FH,"/home/spawn/aa.pl");
$a++ while(<FH>);
print $a;
```

حيث يقوم هذا المقطع البرمجي أعلاه بعمل حساب لعدد الاسطر الموجودة في الملف الذي تم اعطاءه في المقطع أعلاه والصورة الناتجة عن تنفيذ هذا المقطع البرمجي هي كألاتي



Figure (13)
Result

يوضح ال*Figure (13)* ان الملف الذي تم أدخاله في المقطع البرمجي أعلاه هو عبارة عن ملف مكون 26 سطر فقط اي أن هذا الرقم هو رقم يشير الى عدد الاسطر الموجودة في الملف

How To Use Regexp

في بعض الاحيان تكون بعض الملفات التي يتم العمل عليها هي عبارة عن ملفات كبيرة من ناحية عدد الاسطر وعدد الكلمات التي تحتويها لهذا فأذا كان المبرمج في حاجة لأن يقوم بالبحث عن كلمة معينة داخل ملف من هذه الملفات فأن لغة البيرل توفر سهولة البحث عن الكلمة المطلوبة من خلال استخدام تقنية *Regular Expression* او التعابير القياسية او المنتظمة ويوضح المقطع البرمجي ألاتي الكيفية التي تتم بها هذه العملية من خلال لغة البيرل

*Code(18)

```
open (FH,"/home/spawn/aa.pl");
while (<FH>){
if (/perl/){
print  "There is a match","\n";
}
else{
print "There is no match","\n";
}
}
```

واذا تم تنفيذ هذا المقطع البرمجي أعلاه فأن ناتج تنفيذ هذا الكود سوف يكون كألاتي

```
There is no match
There is no match
There is a match
There is no match
```

Figure (14)
RegExp With Files

يوضح*Figure (14)*ناتج تنفيذ المقطع البرمجي أعلاه ويلاحظ على هذا المقطع انه يقوم بعمل فحص كامل على ال
محتوى الملف المعطى في البرنامج وأن محتوى الملف المعطى في البرنامج هو كالأتي

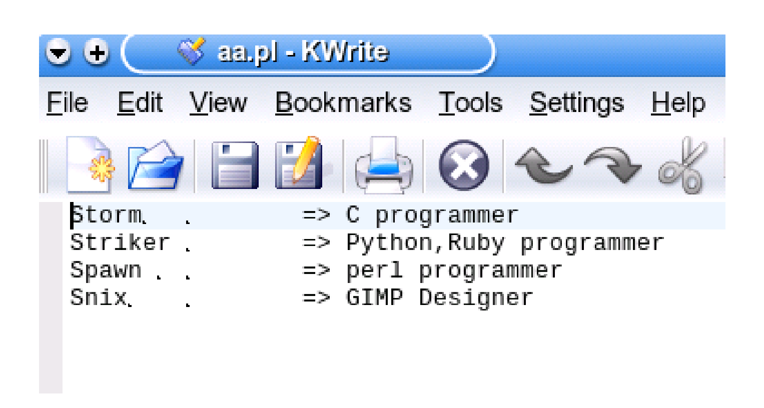


Figure (15)
File Content

وأن اسلوب عمل هذا الكود كان متمثلا بعمل فحص لكافة اسطر التي يحتويها الملف وفي السطر الذي وجد فيه تطابق نفذ البرنامج شرط التطابق وفي الاسطر الاخرى التي لم تحتوي على الكلمة المطلوبة عندها تم طباعة الجملة او الامر البرمجي الذي يشير الى عدم وجود تطابق في السطر

ومن الممكن ان يتم كتابة هذا البرنامج بصيغة اخرى وهي كألاتي

*Code(19)

```
open (FH,"/home/spawn/aa.pl");
while (<FH>){
if ($_ =~ /perl/){
print  "There is a match","\n";
}
else{
print "There is no match","\n";
}
}
```

في كل الحالتين يكون الناتج هو نفس الناتج وللمبرمج حرية اختيار الطريقة الانسب

System Function

من الممكن على المبرمج في لغة البيرل ان يقوم باستعمال الدوال العادية التي تطبق على الملفات في الحالات العادية ومن امثلة هذه الدوال هي

*Code(20)

```
rename
```

*Code(21)

```
chmod
```

يتم تطبيقها في لغة البيرل بنقس الطريقة التي يتم تنفيذها في الحالات العادية

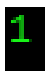
Unlink

هذه الدالة يتم استخدامها من اجل مسح الملفات التي ليس للمستخدم او المبرمج حاجة بها ويكون التمثيل البرمجي الخاص بهذه الدالة كما يلي

*Code()

```
$x=unlink("/home/spawn/bb.pl");
print $x;
```

الان عندما يتم تنفيذ هذا المقطع البرمجي فأن هذا البرنامج سوف يعيد رقم وهذا الرقم المعاد من قبل البرنامج يشير الى عدد الملفات التي تم الغاءها من النظام بنجاح ويوضح ال*Figure(16)*التالي ناتج تنفيذ هذا المقطع البرمجي



Figure(16)
Result

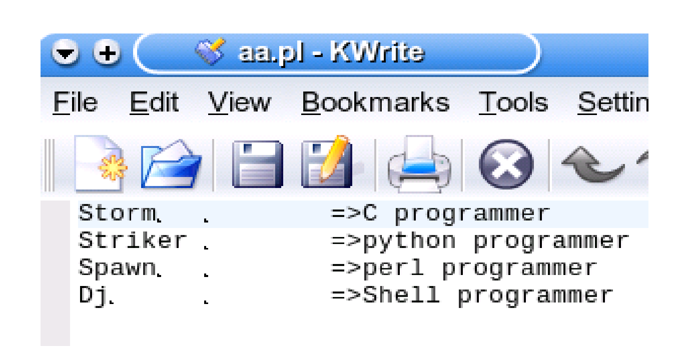
How To Reverse File

ان عملية عكس المحتويات باستخدام دالة ال *reverse* هي عملية مقتصرة على المتغيرات التي تكون من نوع المصفوفات لذا في هذه الحالة اذا كان المبرمج يرغب بأن يقوم بعكس محتويات لملف يجب ان يتم اسناد محتوى الملف الى مصفوفة لكي تتم عملية العكس بصورة صحيحة و المقطع البرمجي ألاتي يوضح الكيفية التي تتم بهذا هذه العملية

**Code(23)*

```
open (FH,"/home/spawn/aa.pl");
while (<FH>){
push (@array,$_);
}
@rev=reverse(@array);
print @rev;
```

المقطع البرمجي ألاتي يوضح الطريقة التي يتم من خلالها عكس محتوى الملف ويوضح ال *Figure (15)* شكل الملف من قبل ان تتم عملية عكس المحتوى عليه



Figure(17)
B4

هكذا يكون شكل الملف قبل ان تتم عملية عكس المحتوى ولكن بعد ان تتم عملية عكس محتوى الملف فأن ناتج تنفيذ المقطع البرمجي المرقم بالرقم *22* سوف تكون كألاتي

```
Dj          =>Shell programmer
Spawn       =>perl programmer
Striker     =>python programmer
Storm       =>C programmer
```

Figure (18)
After

الاساس البرمجي الذي تم أتباعه في هذا المقطع البرمجي كان كما يلي
اولا ان يتم استعمال مصفوفة لكي يتم ادخال المتغير *$_*
اليها من خلال دالة ال *push* على أعتبار ان المتغير أعلاه الذي يمثل النص الموجود في الملف هو عبارة عن متغير عادي
ثانيا يتم الخروج من ال *Loop* الخاصة بال *While*
ثالثا يتم ادخال مصفوفة اخرى تحتوي هذه المصفوفة على الدالة المختصة بعمل ال *reverse* ومن ثم يتم ادخال المصفوفة الاولى التي تم ادخالها في *loop* ال *while*
رابعا يتم طباعة المصفوفة الثانية وعندها يتم عكس الملف

Control Text

في كثير من الملفات وخاصة الملفات المتعلقة بالنظام تكون البنية العامة لهذه الملفات هي كألاتي

```
# /etc/services:
# $Id: services 105871 2005-12-07 06:53:01Z flepied $
#
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, most entries here have two entries
# even if the protocol doesn't support UDP operations.
# Updated from RFC 1700, ``Assigned Numbers'' (October 1994).  Not all ports
# are included, only the more common ones.
#
# The latest IANA port assignments can be gotten from
#       http://www.iana.org/assignments/port-numbers
# The Well Known Ports are those from 0 through 1023.
# The Registered Ports are those from 1024 through 49151
# The Dynamic and/or Private Ports are those from 49152 through 65535
#
# Each line describes one service, and is of the form:
#
# service-name  port/protocol  [aliases ...]   [# comment]

tcpmux        1/tcp                          # TCP port service multiplexer
tcpmux        1/udp                          # TCP port service multiplexer
rje           5/tcp                          # Remote Job Entry
rje           5/udp                          # Remote Job Entry
echo          7/tcp
echo          7/udp
discard       9/tcp           sink null
discard       9/udp           sink null
systat        11/tcp          users
systat        11/udp          users
```

Figure (19)
File Content

المقصود من الشكل أعلاه هو ان الملفات هذه تحتوي على العديد من التعليقات و ال *useless comment* لذا لو المبرمج في يرغب بأن يقوم باستخلاص ال *gist* من الملف المقطع البرمجي ألاتي يقوم بهذه العملية

**Code (24)*

```
open (FH,PATH");
while (<FH>){
  chomp
    s/#.*//;
print $_;
}
```

ففي هذه الحالة عندما يتم تنفيذ هذا المقطع البرمجي على الملف المحدد في البرنامج يتم استعمال هذا البرنامج والسطر الثالث

*Code (25)

```
chomp
```

يتم استخدام هذا الملف من أجل الغاء ال *new line* في الملفات
السطر الرابع

*Code (26)

```
s/#.*//;
```

يتم من اجل الغاء التعليقات او ال *comments* الموجودة في الملف الذي يتم العمل عليه

The Modules Of The Files

ان المفات في لغة البيرل التي تعتبر من الاركان الاساسية التي توفر الموديلات البرمجية مرونة كبيرة في التعامل معها و هناك العديد من الموديلات البرمجية في لغة البيرل التي تتناول برمجة الملفات و اسلوب التعامل معها وفي هذا الجزء سوف يتم التطرق الى اهم الموديلات التي تتعامل مع الملفات واسلوب البرمجة بها واستثمار الامكانيات التي تحتويها للحصول على أكبر قدر من الراحة في التعامل مع الملفات

File::Basename

1-basename

يحتوي هذا الموديل على دالتين لها عمل مهم و الدالة الاولى التي يحتويها هذا الملف هي الدالة *basename*وهذه الدالة اذا تم أدخالها في كود برمجي تعمل على ايجاد الاسم الاخير للملف الذي يتم التعامل معه ويكون التمثيل البرمجي الخاص بها كما يلي من خلال هذا المقطع البرمجي

*Code (27)

```
use File::Basename;
$a="/home/spawn/aa.pl";
$b=basename($a);
print $b;
```

يعمل المقطع البرمجي أعلاه على ايجاد اسم الملف الذي يتم التعامل معه في البرنامج وناتج تنفيذ هذا المقطع البرمجي هو كألاتي

```
aa.pl
```

Figure (20)
basename

dirname

اما الدالة الاخرى التي يحتويها هذا الموديل هي دالة ال *dirname* هي من ناحية العمل مشابه لما تقوم به الدالة السابقة ولكن هذه الدالة تختلف عن الدالة السابقة انها تقوم بأعطاء اسم المجلد الذي يحتوي على هذا الملف ويكون التمثيل البرمجي لهذه الدالة نفس التمثيل البرمجي للدالة السابقة

*Code (28)

```
use File::Basename;
$a="/home/spawn/aa.pl";
$b=dirname($a);
print $b;
```

ناتج تنفيذ هذا المقطع البرمجي هو كألاتي

/home/spawn

Figure(21)
Dirname

File::Copy

يعتبر هذا الموديل من الموديلات الاساسية التي تقدم اساسيات وخواص الانتقال و التنقل بين الملفات ويحتوي هذا الموديل على دوال منها

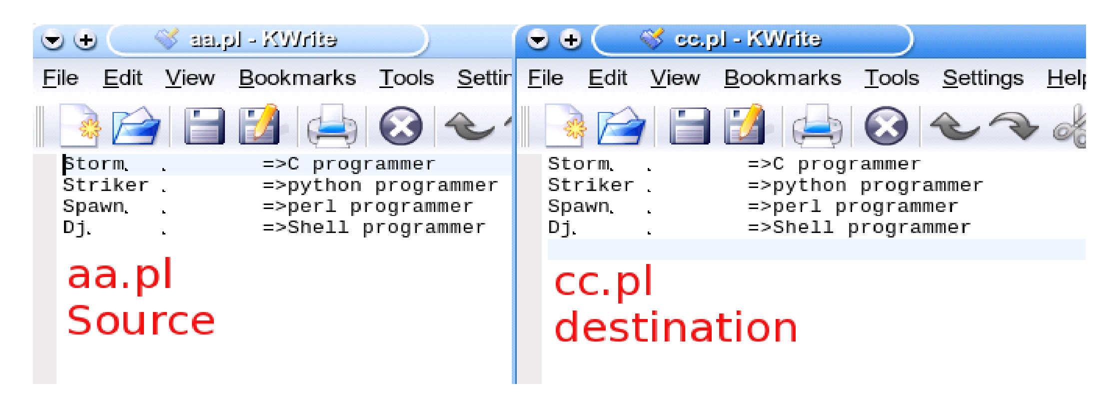
copy

يكون عمل هذه الدالة مختصرا على النسخ كما هو واضحا من الاسم ولكن هذا النسخ يكون معتمد على محتوى الى الملف اي ان هذه الدالة تعمل على نسخ محتوى الملف من ملف ال *source* الى ملف ال *destination*ويكون التمثيل البرمجي لهذه الدالة كما يلي في هذا المقطع البرمجي

*Code (29)

```
use File::Copy;
copy ("/home/spawn/aa.pl","/home/spawn/cc.pl");
```

الان عندما يتم تنفيذ المقطع البرمجي أعلاه عندها سوف يتم نسخ محتويات الملف الاول والذي يحمل الاسم *aa.pl* الى الملف الثاني الذي يحمل الاسم *cc.pl*وتجدر الاشارة الى ان هذا الموديل له القدرة على عمل ال *create* اي انه اذا كان ملف ال *destination* غير موجود في المكان الذي ستتم عملية النسخ اليه ففي هذه الحالة سوف هذا الموديل بتوليد هذا الملف لكي تتم عملية النسخ و ال*Figure(19)*يوضح الطريقة التي تتم بها عملية النسخ



Figure(22)
Copy (source,destination)

cp

هذه الدالة من الدوال التي يتم استعمالها في عمليات النسخ و هذه الدالة تعمل على نسخ الملف وليش نسخ محتواه كما كانت تفعل الدالة السابقة ويكون التمثيل البرمجي لهذه الدالة كما يلي

*Code (30)

```
use File::Copy "cp";
cp ("/home/spawn/aa.pl","/home/spawn/Desktop");
```

اما هذه الدالة فأن العمل الذي تقوم به هو عمل نسخ صرف اي تقوم بنسخ الملف aa.pl الى المسار المحدد في في الجزء الثاني من المسار فقط

Move

اما عن هذه الدالة فأن عملها يكون اشبه بعمل الامر *cut* الذي يعمل على قص الملف من المكان الموجود فيه ولصقه في مكان اخر من دون الاحتفاظ بنسخة من الملف في المكان الذي تم قصها منه ويكون التمثيل البرمجي لهذه الدالة كما يلي من خلال هذا المقطع البرمجي

*Code (31)

```
use File::Copy;
move ("/home/spawn/cc.pl","/home/spawn/Desktop");
```

File::Temp

يعتبر هذا الموديل من افضل الموديلات البرمجية التي يتم استخدامها في عملية التعامل مع ملفات ال *tmp* ومن اهم الدوال التي يحتويها هذا الموديل هي دالة *File::Temp* حيث تقوم هذه الدالة بالتعامل مع هذا النوع من الملفات التي تكون من نوع ملفات ال *temp* ويكون اسلوب التمثيل البرمجي لها بالشكل ألاتي

*Code (32)

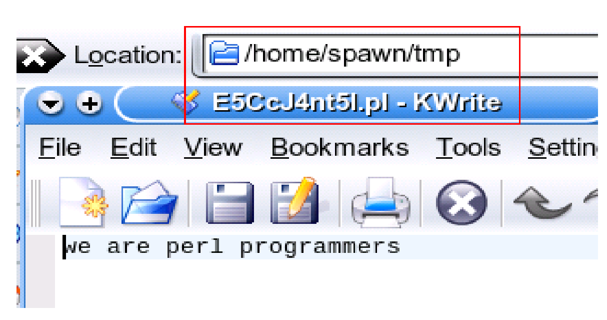
```
use File::Temp;
 $tmp = new File::Temp( UNLINK => 0, SUFFIX => '.pl');
 print $tmp "we are perl programmers\n";
 print "Filename is $tmp\n";
```

هذه هي الطريقة العامة التي يتم من خلالها توليد الملفات المؤقتة في لغة البيرل و الان اذا تم تنفيذ هذا المقطع البرمجي فأن الناتج من عملية التنفيذ ةسوف تكون كما يلي بال*Figure(20)*

```
Filename is /home/spawn/tmp/E5CcJ4nt5l.pl
```

Figure(23)
Temp File

ويوضح الشكل الذي في الاعلى المسار الذي تم توليد الملف المؤقت فيه اي في مجلد ال *tmp*في مسار ال *home* الخاص بال *current user* و الفعل البرمجي على هذا الملف يكون كألاتي



Figure(24)
Path & Content of (Temp File)

*Code (33)

```
SUFFIX => '.pl'
```

هذه الجزئية تكون مسؤولة عن امتداد الملف فهذا يعني ان الملف المؤقت الذي سوف يتم انشاءه و التعامل معه سوف يكون ملف له أمتداد *pl*.وكيف ما يتم تغيير هذا الملحق سوف يكون امتداد الملف مطابق لهذا الملحق ويتيح هذا الموديل ايضا امكانية تحديد المسار الذي سوف يتم توليد المؤقت فيه وفق رغبة المبرمج وحسب المكان الذي يرغب ان يولد الملف المؤقت فيه وتتم هذه الطريقة كما يلي من خلال هذا المقطع البرمجي التالي

*Code (34)

```
use File::Temp;
 $tmp = new File::Temp( UNLINK => 0, SUFFIX => '.pl',
                                         DIR =>"/home/spawn/Desktop");
 print $tmp "we are perl programmers\n";
 print "Filename is $tmp\n";
```

الان لو يتم تنفيذ هذا المقطع البرمجي فان الناتج من عملية تنفيذه سوف يكون كألاتي من خلال ال*Figure(21)*

```
Filename is /home/spawn/Desktop/h5YngLB9FW.pl
```

Figure(25)
Result With Specific Dir

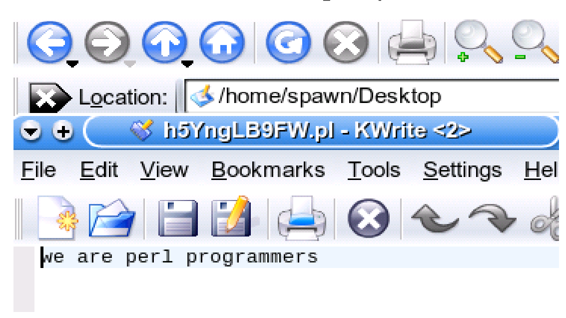


Figure (26)
Result Of Execution

How To Stat

هذا الموديل البرمجي يقوم بعمل بسيط ولكن مهم في نفس الوقت هذا الموديل يكون مسؤول عن اعطاء معلومات عن حالة الملف الملف الذي يتم التعامل معه في المقطع البرمجي ويكون التمثيل البرمجي العام لهذا الموديل كما يلي من خلال هذا المقطع البرمجي

*Code (35)

```
use File::stat;
$perl = stat("/home/spawn/aa.pl");
print $perl->size,"\n";
print $perl->atime,"\n";
print $perl->mtime,"\n";
print $perl->uid,"\n";
print $perl->gid,"\n";
print $perl->mode,"\n";
print $perl->nlink,"\n";
```

والان اذا تم تنفيذ هذا المقطع البرمجي فأن الناتج من عملية التنفيذ هذه سوف يكون كما يلي من خلال Figure(23)

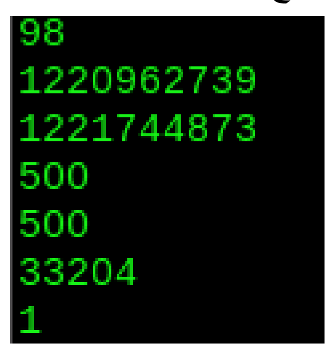


Figure (27)
File Stat

ان ناتج تنفيذ هذا الكود هو عبارة عن ارقام وهذه الارقام هي عبارة عن ارقام تعطي حالة تفصيلية لحالة الملف الذي يتم العمل عليه في المقطع البرمجي أعلاه و هذه الدوال التي تم استخدامها في المقطع البرمجي فهي تعني ألاتي

1- size

ان الناتج من هذه الدالة هو عبارة عن رقم يشير الى حجم الملف الذي يتم التعامل معه في المقطع البرمجي و هذا الرقم يكون معنون بصيغة البت و هذا يعني ان حجم الملف الذي يحمل الاسم *aa.pl* يبلغ من الحجم *98* بت

aa.pl (98 B) Perl Program

Figure (28)
File Size

2- atime

هذه الدالة يتم استعمالها مع الملف لكي يتم معرفة *Last access time* حيث من خلال هذه الدالة يتم معرفة هذه الخاصية

3- mtime

هذه الدالة هي الدالة التي يتم استخدامها من أجل معرفة ال *Last modified time*

4- uid

هذه الدالة يتم استخدامها من اجل ان يتم معرفة ال *uid* الخاصة بالملف الذي تم التعامل معه في المقطع البرمجي

5- gid

هذه الدالة التي يتم التعامل معها من اجل معرفة ال *Gid* الخاصة بالملف الذي يتم التعامل معه في المقطع البرمجي

6- mode

هذه الدالة التي يتم التعامل معها من اجل معرفة **نمط الملف** الذي يتم التعامل معه في المقطع البرمجي

7-nlink

هذه الدالة التي يتم التعامل معها من أجل معرفة ال *number of links* للملف الذي يتم التعامل معه في المقطع البرمجي

How To Find Files

File::Find

يتم استخدام الموديل البرمجي من أجل ان يتم البحث عن الملفات في لغة البيرل

Find

هذه الدالة التابعة لهذا الموديل تقوم بمهمة البحث عن الملفات المطلوبة لمسار محدد مسبقا من قبل المبرمج ويكون التمثيل العام لهذا الموديل كما يلي في المقطع ألاتي

*Code (36)

```
use File::Find;
find sub { print "$File::Find::name\n" if -f }, "/home/spawn/";
```

هذا المقطع البرمجي هو البرنامج المسؤول عن البحث عن الملفات في مسار ال */home/spawn*

*Code (37)

```
if -f
```

هذه الجزئية من المقطع البرمجي تكون مسؤولة عن طباعة المسار اذا كان الملف الذي نبحث عنه هو **ملف فعلي** اي بعبارة اخرى الناتج من عملية الطباعة ملفات فقط اي المجلدات لن يتم عرضها في ناتج البرنامج هذا المقصود بعبارة الملف الفعلي ولهذا تم ادراج هذه الفقرة في المقطع البرمجي وللمبرمج حرية الاختيار بين وضعها في البرنامج او رفعها منه

أما الناتج من تنفيذ المقطع البرمجي أعلاه فأنه يكون كألاتي

```
/home/spawn/tmp/kde-spawn/kio_chme2vKvb.html
/home/spawn/tmp/kde-spawn/kio_chmYjEy3b.html
/home/spawn/tmp/kde-spawn/konquerorlShQ7a.tmp
/home/spawn/tmp/kde-spawn/kio_chmp1acDa.html
/home/spawn/tmp/kde-spawn/kio_chm6WX6bc.html
/home/spawn/tmp/kde-spawn/kio_chmMcaoKa.html
/home/spawn/tmp/kde-spawn/konquerorz35waa.tmp.part
/home/spawn/tmp/kde-spawn/kio_chmraMNdb.html
/home/spawn/tmp/kde-spawn/kio_chmxPcarc.html
/home/spawn/tmp/kde-spawn/kio_chmhbvb0a.html
/home/spawn/tmp/kde-spawn/kio_chm8Azbzb.html
/home/spawn/tmp/kde-spawn/kio_chmz2YEeb.html
```

Figure (29)
Part Of Find Result

How To Compare

File::Compare

تتيح لغة البيرل كافة المبرمجين القيام بعمليات المقارنة بين الملفات التي يتم التعامل معها وهذه العملية من الممكن ان يتم القيام بها من خلال الاستعانة بالموديل الذي يحمل الاسم *File::Compare* لكي يتسنى للمبرمج ان يميز الفرق بين الملفات التي يتعامل معها
ويكون التمثيل البرمجي لعملية المقارنة بين الملفات كما يلي من خلال المقطع البرمجي ألاتي

*Code (38)

```
use File::Compare;
      if (compare("/home/spawn/aa.pl","/home/spawn/ss.pl") == 0) {
         print "They're equal\n";
      }
else {
print "Not Equal\n";
}
```

وضح المقطع البرمجي أعلاه الطريقة التي تمت بها المقارنة بين الملفات ولكن تجدر الاشارة الى انه اسلوب المقارنة بين الملفات لايكون معتمد على اسم الملف ولكن معتمد على محتوى الملف يعني اذا كان محتوى الملف الاول مطابق لمحتوى الملف الثاني فهذا يعني انه عملية المقارنة تشير الى وجود تطابق اما اذا كان المحتوى مختلف فأن عملية المقارنة سوف تشير الى وجود اختلاف بين محتوى الملف الاول ومحتوى الملف الثاني

```
They're equal
```

Figure(A) (30)
Equal

```
Not Equal
```

Figure (B) (30)
Not Equal

يشير الشكل الاول الى وجود حالة تطابق بين الملفين الذين تمت المقارنة بينهما وكان محتواهما متطابق لذا تم الحصول على علامة التطابق ومن ثم في الشكل الثاني تم التلاعب بمحتوى احد الملفين بحث اصبح محتواهما مختلف عندها تم الحصول على اشارة عدم جود تطابق بين الملفين

List::Util

يحتوي هذا الموديل البرمجي على عدد من العمليات التي تمكن المستخدم على التعامل بحرية أكبر مع الملفات في لغة البيرل وتتيح للمبرمج ان يكون على قدر أكبر من السيطرة على ما تحتويه الملفات من معلومات وبيانات وكيفية التعامل معها

1-shuffle

هذه الدالة تتعامل مع الملف المعطى لها بصورة عادية ولكنها تعمل على أعادة محتويات الملف الى المستخدم بصورة عشوائية وغير منتظمة يكون التمثيل البرمجي لهذه الدالة كما يلي من خلال هذا المقطع البرمجي

*Code (39)

```
use List::Util qw(shuffle);
open(FH,"/home/spawn/aa.pl");
while (<FH>){
push (@lines,$_);
}
@lines=shuffle(@lines);
print @lines;
```

اما الان عندما يتم تنفيذ هذا الكود البرمجي فأن النتاتج من عملية التنفيذ سوف يكون كما يلي من خلال الشكل التوضيحي الذي يوضح ناتج التنفيذ

```
Storm          =>C programmer
Striker        =>Python programmer
Dj             =>Shell programmer
Spawn          =>Perl programmer
```

Figure (31)
Result Of Shuffling

ولكن تجدر الاشارة الى ان عملية التوليد العشوائية التي تمت على الملف في المقطع البرمجي أعلاه لا يؤثر على محتوى الملف اي لاتتم عملية اعادة ترتيب له كما في ناتج التنفيذ بل سوف يبقى كما تمت كتابته في المرة الاولى

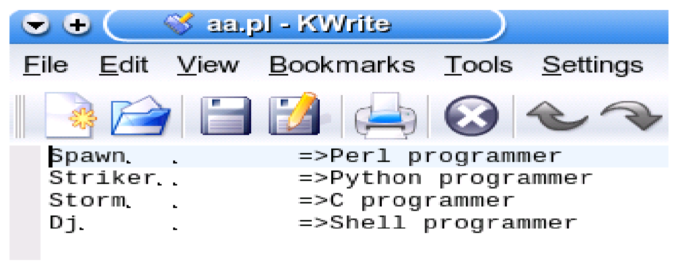


Figure (32)
The Original Content

2- maxstr

هذه الدالة تقوم بقراءة محتويات الملف سطر بسطر ومن ثم بعد ذلك عند الانتهاء من عملية القراءة الخاصة بالملف تقوم هذه الدالة بأعادة أكبر قيمة موجودة في الملف يكون التمثيل العام لهذه الدالة كما يلي من خلال المقطع البرمجي أدناه

*Code (40)

```
use List::Util qw(maxstr);
open(FH,"/home/spawn/aa.pl");
while (<FH>){
push (@lines,$_);
}
@lines=maxstr(@lines);
print @lines,"\n";
```

الان عندما يتم تنفيذ المقطع البرمجي أعلاه ويتم قراءة محتويات الملف فأن البرنامج سوف يعيد في هذه الحالة السطر الذي يحمل أكبر قيمة حرفية اي بعبارة اخرى ان السطر الذي سيعاد هو السطر الذي يحتوي على أكبر عدد من الحروف فيه

```
Striker            =>Python programmer
```

Figure (33)
Max String

هذه كانت في حالة الاحرف اما في حالة الارقام فالامر سيان سوف يتم أعادة الرقم الأكبر حتى وان كانت الارقام مكونة من نفس عدد المراتب كالتالي

*Code (41)

```
use List::Util qw(maxstr);
open(FH,"/home/spawn/ss.pl");
while (<FH>){
push (@lines,$_);
}
@lines=maxstr(@lines);
print @lines,"\n";
```

وناتج التنفيذ من هذا المقطع البرمجي هو الناتج ألاتي

```
13
```

Figure (34)
Max Str

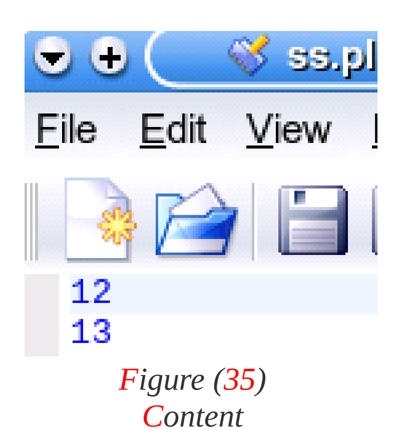


Figure (35)
Content

يوضح الFigure(35) ان الدالة البرمجية maxstrحتى مع الارقام المتساوية من ناحية عدد المراتب تعمل مقارنة بين هذه الاعداد وتعيد العدد الأكبر بينها

3-minstr

يلاحظ من اسم الدالة أعلاه انها تقوم بعمل مشابه لعمل الدالة السابقة ولكن بصورة معكوسة حيث ان هذه الدالة تقوم بقراءة الملف المعطى لها في المقطع البرمجي ومن ثم تعمل على اعادة السطر الاقصر اي السطر الذي يحتوي على العدد الاقل من الاحرف والرموز ويكون التمثيل البرمجي العام لهذه الدالة هو كما يلي

*Code (42)

```
use List::Util qw(minstr);
open(FH,"/home/spawn/aa.pl");
while (<FH>){
push (@lines,$_);
}
@lines=minstr(@lines);
print @lines,"\n";
```

الان عندما يتم تنفيذ هذا المقطع البرمجي فأن الناتج من عملية التنفيذ هذه سوف تكون كألاتي من خلال الشكل التوضيحي أدناه



Figure(36)
Minstr

هذا يعني ان السطر الذي يحتوي على اقل عدد من الاحرف والرموز هو السطر الموضح في الFigure (36)

FileHandle

1- open

يوفر هذا الموديل دالة داخلية فيه هي دالة ال *open* وهذه الدالة كما هو واضح من الاسم فهي تقوم بفتح الملف المعطى لها في المقطع البرمجي من ثم تقوم بطباعة محتويات هذا الملف يكون التمثيل البرمجي العام لهذه الدالة كما يلي

*Code (43)

```
use FileHandle;
   $fh = new FileHandle;
   if ($fh->open("/home/spawn/aa.pl")) {
      print <$fh>;
      $fh->close;
   }
```

وعندما يتم تنفيذ هذا المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ هذه يكون كالأتي من خلال الشكل أدناه

```
Spawn           =>Perl programmer
Striker         =>Python programmer
Storm           =>C programmer
Dj              =>Shell programmer
```

Figure (37)
Open Handle

2- write

هذا ليس الاسم الرسمي لهذه الدالة ولكن هذا هو العمل الذي تقوم به بالتحديد حيث انها تقوم بارسال بيانات الى الملف المعطى في المقطع البرمجي ومن ثم تقوم بعملية الكتابة على الملف ويكون التمثيل البرمجي لها كام يلي من خلال هذا المقطع

*Code (44)

```
use FileHandle;
 $fh = new FileHandle "> /home/spawn/ss.pl";
   if (defined $fh) {
      print $fh "We are perl programmers\n";
      $fh->close;
   }
```

اسلوب عمل هذا البرنامج يكون كما يلي يتم اسناد الكائن الذي يحمل الاسم *fh*الى مسار ملف ما من المفترض لهذا

الملف ان يكون موجود فعلا على النظام ومن ثم يتم استدعاء دالة التعريف حيث اذا كان المسار للملف المعطى هو فعلا معرف عندها سوف يتم تنفيذ الشرط وتنفيذ جملة الطباعة الى الملف المعطى ويكون ناتج تنفيذ البرنامج كما يلي من خلال هذا الشكل أدناه

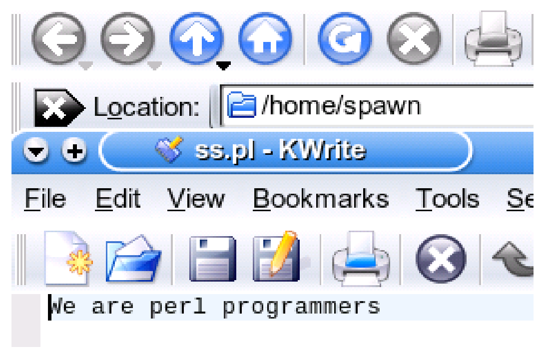


Figure (38)
Write FileHandle

Tie::File

هذا الموديل يقوم بالتعامل مع الملف المعطى بالمقطع البرمجي على انه مصفوفة وكافة العمليات التي تتم على المصفوفة من الممكن ان تتم على الملف لانه محتوى الملف اصبح الان مصفوفة ويكون التمثيل البرمجي لهذا الموديل هو كما يلي

*Code (45)

```
use Tie::File;
tie @array,'Tie::File',"/home/spawn/aa.pl";
print @array;
```

والان اذا تم تنفيذ هذا المقطع البرمجي فأن الناتج من عملية التنفيذ هذه سوف يكون كألاتي من

```
Spawn          =>Perl programmer Striker          =>Python programmer Storm          =>C programmer Dj          =>Shell pro
grammer
```

Figure (39)
Result Of Reading

وان سبب ظهور ناتج عملية الطباعة بهذه الصورة فان السبب في هذا هو أنه وحتوى ملف قد اصبح الان مصفوفة وعند طباعة المصفوفة فأن ناتج عملية الطباعة يكون كما في Figure 39

Printing Single Line

ولكن بما أنه محتوى الملف الان قد اصبح مصفوفة اذن في هذه الحالة من الممكن لن تتم طباعة اسطر الملف كل سطر بمفرده وذلك على اعتبار ان اسطر الملف هي عناصر المصفوفة وهذه العملية تتم من خلال المقطع البرمجي ألاتي

*Code (46)

```
use Tie::File;
tie @array,'Tie::File',"/home/spawn/aa.pl";
print $array[2],"\n";
```

الان اذا لو تم تنفيذ هذا المقطع البرمجي أدناه فان الناتج من عملية التنفيذ هي ان يتم طباعة السطر الثالث من الملف الموجود في المقطع البرمجي

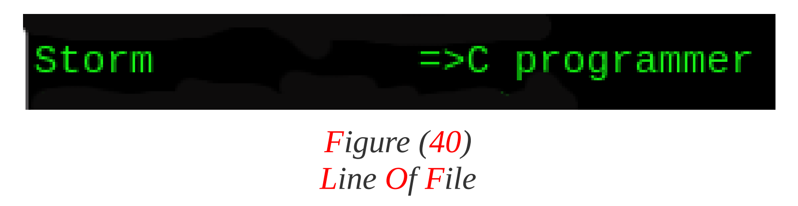


Figure (40)
Line Of File

How To Assign

من الممكن ان يتم التعامل مع الملف على انه متغير من نوع مصفوفة اذن في هذه الحالة من الممكن ان يتم اضافة أسطر جديدة الى الملف وتغيير القيم الخاصة بالاسطر وذلك ايضا على اعتبار انها عناصر خاصة بالمصفوفة ومن الممكن ان يتم تمثيل كل من عملية سطر جديد الى الملف
والتغيير على قيمة موجودة مسبقا على الملف من خلال المقطع البرمجي ألاتي الذي يوضح كل من الطريقتين

*Code (47)

```
use Tie::File;
tie @array,'Tie::File',"/home/spawn/aa.pl";
$array[2] = "Storm    =>C Fr34k";
$array[4] = "Snix      => Designer";
print $array[2],"\n";
print $array[4],"\n";
```

أما الان سيتم الاطلاع على محتوى الملف قبل ان يتم تنفيذ المقطع البرمجي أعلاه لكي يكون الفرق واضح على محتوى الملف قبل وبعد التنفيذ

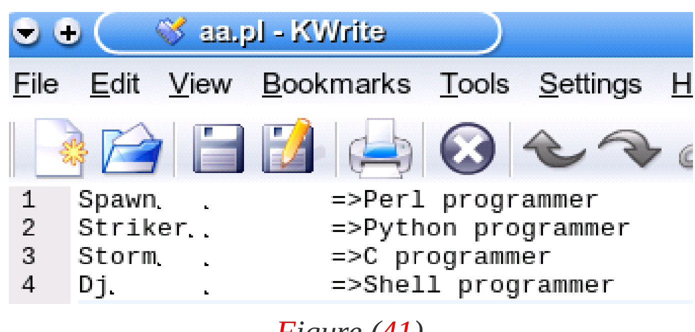


Figure (41)
B4 Exec

اما الان عندما يتم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ سوف تكون كما يلي

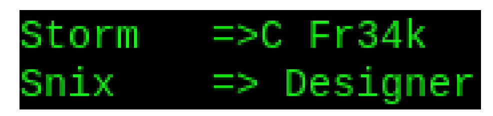


Figure (42)
Result

اما الان شكل الملف عندما يتم تنفيذ المقطع البرمجي سوف يكون كالاتي

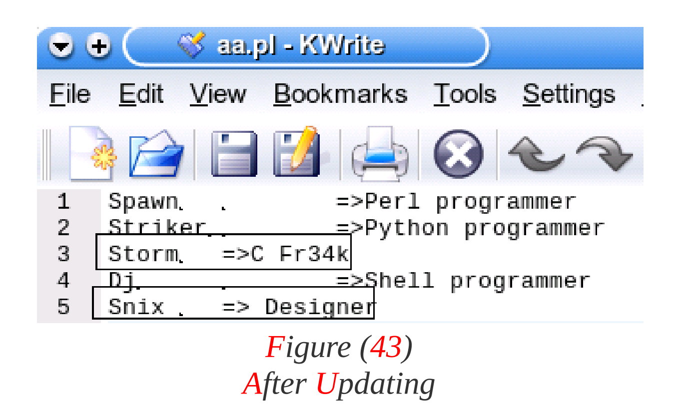


Figure (43)
After Updating

IO::File

هذا الموديل البرمجي ايضا يتعبر من الموديلات البرمجية التي توفر أمكانية التحكم بالملفات اي من قابلية فتح الملفات وطباعة محتوياتها والكتابة على الملفات وانشاء الملفات الخ

How to open

لهذا الموديل الامكانية على فتح الملفات و التعامل معها وان فتح الملفات وطباعة ما تحتويه من بيانات ومعلومات يكون التمثيل البرمجي العام لها كما يلي من خلال المقطع البرمجي أدناه

**Code (48)*

```
use IO::File;
  $fh = new IO::File;
  if ($fh->open("/home/spawn/aa.pl")) {
     print <$fh>;
  }
```

والان لو تم تنفيذ المقطع أعلاه فأن الناتج من عملية التنفيذ سوف يكون كالاتي

```
Spawn            =>Perl programmer
Striker          =>Python programmer
Storm    =>C Fr34k
Dj               =>Shell programmer
Snix     => Designer
```

Figure (44)
IO Output

Write & Creat

هذه الجزئية من هذا الموديل تقوم بعملين بنفس الوقت حيث تكون الجزئية الاولى هي الجزئية المسؤولة عن الكتابة على الملفات ويكون التمثيل البرمجي لها كما يلي في المقطع ألاتي

*Code (49)

```
use IO::File;
   $fh = new IO::File ">/home/spawn/ss.pl";
   if (defined $fh) {
      print $fh "We are perl programmers\n";
   }
```

الان لو يتم تنفيذ المقطع البرمجي أعلاه فان الناتج من عملية تنفيذ هذا الكود سوف تكون كألاتي

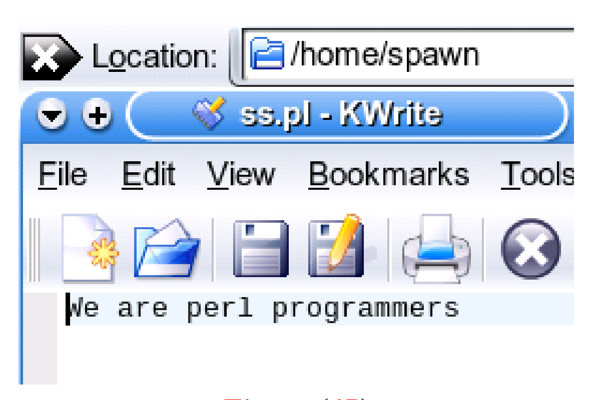


Figure (45)
IO::File Write

كما سبق الذكر هذه الجزئية هي الجزئية التي تكون مسؤولة عن الكتابة على الملفات اذا كانت الملفات مسبقا موجودة اي اذا كانت الملفات معرفة على النظام
الجزئية الثانية هي تتضمن الجزئية الاولى وهي الكتابة على الملفات ولكن اذا لم تكن الملفات موجودة على النظام هذه العملية تتم كما في الطريقة السابقة بدون اي تغيير فهذا يعني ان هذا الموديل له القدرة على انشاء الملفات التي لم تكون موجودة على النظام من قبل و العملية تمثل كما يلي

*Code (50)

```
use IO::File;
   $fh = new IO::File ">/home/spawn/Perl.pl";
   if (defined $fh) {
      print $fh "We are perl programmers\n";
   }
```

الان اذا تم تنفيذ المقطع البرمجي أعلاه فان الناتج من هذه العملية هو تكوين ملف يحمل الاسم *Perl.pl* في مجلد المستخدم *spawn* ويوضح ال *Figure(46)* ناتج التنفيذ

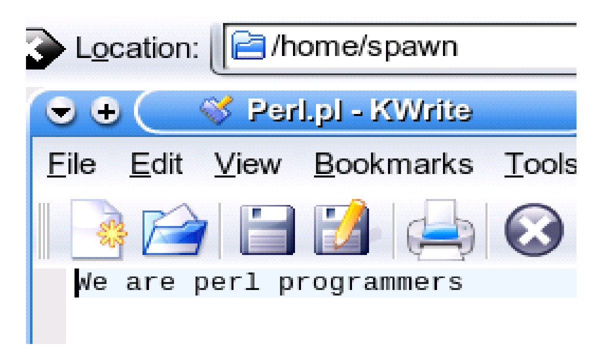


Figure (46)
IO::File Write & Create

Regular Expression (RegExp)

التعابير القياسية او التعابير المنتظمة في لغة البيرل تعتبر التعابير المنتظمة واحدة من اهم الخواص والامكانيات التي منحت هذه اللغة الشهرة و القوة التي تتمتع بها وفي بعض الاحيان يتم اطلاق مصطلح *pattern matching* على التعابير القياسية ولكن كل من التعبير الاول و التعبير الثاني لهم نفس الدلالة وتعتبر تقنية التعابير القياسية من اوسع وأكبر التقنيات في لغة البيرل لما تحتويه من خيارات و اوامر عدة تؤدي عدد من الوظائف
تكون الفكرة العامة للتعابير القياسية هي القيام بعملية مطابقة بين السلسلة النصية والتعبير القياسي واذا تم حصول التطابق سوف يتم تنفيذ شرط معين واذا لم يحصل تطابق سوف يتم تنفيذ شرط أخر
التعبير القياسي هو كل ما حصر بين علامات ال

```
/ RegExp   /
```

Figure (1)
RegExp

يكون التمثيل البرمجي العام للتعابير القياسية
في لغة البيرل كما يلي من خلال المقطع البرمجي أدناه و الذي يعبر عن التعابير القياسية بابسط صورة

**Code(1)*

```
$a="perl programming";
$b="perl ";
if ($a =~ /$b/){
print "Ok match\n";
}
else {
print "Sorry ,No match";
}
```

الان عندما يتم تنفيذ المقطع البرمجي أعلاه فان الناتج من عملية التنفيذ تكون كما يلي من خلال الشكل ألاتي

```
Ok match
```

Figure (2)
Result

وتوجد طريقة اخرى يتم من خلالها تمثيل التعابير القياسية وهذه الطريقة الجديدة سوف تعطي نتيجة مشابهة للنتيجة التي يتم الحصول عليها من التمثيل السابق وهذه الطريقة تكون كما يلي من خلال المقطع البرمجي ألاتي

**Code(2)*

```
$_="perl programming";
if (/perl/){
print "Ok,Match\n";
}
else {
print "No,Match\n";
}
```

والان اذا تم تنفيذ الكود أعلاه فأن الناتج من عملية التنفيذ هو الناتج ذاته من المقطع البرمجي الذي تم تنفيذه في السابق

```
Ok match
```

Figure (2)
Result

الان اذا تم عمل *Over view* على كل من المقطعين الاول و الثاني فان القيمة التي تكون مسندة الى المتغير

**Code(3)*

```
$a="perl programming";
```

**Code(4)*

```
$_="perl programming";
```

هي عبارة عن سلاسل نصية عادية *Simple Perl Strings* اما عن الكلمات التي تكون محصورة بين علامات

**Code(5)*

```
/AAA/
```

هي التعابير القياسية و التي تتم بها على اثرها المطابقة حيث من خلالها يتم معرفة هل يوجد حالة تطابق في البرنامج ام لايوجد حالة تطابق

No White Space Allowed

اثناء علميات المقارنة في لغة البيرل على المبرمج ان يكون حذرا من وجود ال *White Spaces* في البرنامج لانه في هذه الحالة سوف لن تتم عملية المقارنة

**Code(6)*

```
$a="perl";
$b="perl ";
if ($a =~ /$b/){
print "Ok Match\n";
}
else {
print "Bad No Match\n";
}
```

الان لو تم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ سوف يكون كما يلي من في الشكل التوضيحي

```
Bad No Match
```

Figure (3)
WhiteSpaces

يلاحظ على هذا البرنامج انه يحتوي على متغيرين وكل من هذين المتغيرين هم متغيرات متشابهة ومتطابقة الى حد كبير ولكن في المتغير الثاني يوجد اختلاف هو وجود فراغ وهذا الفراغ هو فراغ غير موجود في المتغير الاول عند هذه النقطة اصبح الفرق موجود في البرنامج وعند التنفيذ سوف يحدث اختلاف في التطابق

The letter Case

في هذا المرحلة من عمليات التطابق باستخدام التعابير القياسية في لغة البيرل على المبرمج ان يكون قادرا على ان يكون على اطلاع بأنه حالة الاحرف في لغة البيرل لها تأثير على مسار البرنامج وهذا يعني ان الاحرف التي تكون في التشكيل الاعلى تختلف عن حالة الاحرف التي تكون في حالة التشكيل الادني والمقطع البرمجي ألاتي يوضح الطريقة العامة لتمثيل هذه الحالة

*Code(7)

```
$a="perl";
$b="PERL";
if ($a =~ /$b/){
print "Ok Match\n";
}
else {
print "Bad No Match\n";
}
```

الان عندما يتم تنفيذ هذا البرنامج فأن النتيجة التي يتم الحصول عليها هي هي نتيجة تشير على عدم حصول تطابق بين المتغير و التعبير القياسي وذلك بسبب التخالف في حالة الاحرف وهذا يعني انه لغة البيرل لاتعمل على المطابقة بين حالة الاحرف الكبيرة و الحالة الاحرف الصغيرة و وناتج تنفيذ هذا البرنامج هو كألاتي

```
Bad No Match
```

Figure (4)
No LetterCase

No NewLine

في التعابير القياسية في لغة البيرل عندما يكون المبرمج بصدد القيام بعملية مطابقة بين شيئين ففي هذه الحالة يجب اخد ال Escape Sequence في نظر الاعتبار اي انه عملية المطابقة لن تتم في حالة وجود رموز الهروب ويوضح المقطع البرمجي ألاتي المقصود بوضع رموز الهروب في عمليات المطابقة

*Code(8)

```
$a="perl";
$b="PERL\n";
if ($a =~ /$b/){
print "Ok Match\n";
}
else {
print "Bad No Match\n";
}
```

الان عندما يتم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ سوف يكون كما يلي من خلال الشكل أدناه

```
Bad No Match
```

Figure (5)
No NewLine

The Numbers

من الممكن ايضا ان تتم عمليات المقارنة بين الارقام في لغة البيرل و المقطع البرمجي ألاتي يوضح الطريقة العامة التي يتم بها التمثيل البرمجي العام لبرمجة الارقام في التعبير القياسية

**Code(9)*

```
$a="123456789";
$b="123";
if ($a =~ /$b/) {
print "Ok";
}
else {
print "Bad";
}
```

من كل من الناحية النظرية و الناحية البرمجية هذا الاسلوب الخاص لتمثيل الارقام في التعابير القياسية هو اسلوب مشابه جدا لاسلوب تمثيل الاحرف وان الناتج من عملية تنفيذ المقطع أعلاه تكون كالأتي من خلال الشكل التوضيحي



Figure (6)
Numbers

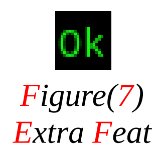
How To Extra

التعابير القياسية في لغة البيرل تمكن المستخدم من قابلية التخلي عن استعمال الارقام الكبيرة و الكثيرة في عمليات المقارنة لذا فأن البيرل في هذه الحالة توفر للمبرمج *Extra Feature* تعرف هذه الخاصية بال *Square Bracket* ويكون التمقيل البرمجي العام لهذه الخاصية كما يلي من خلال المقطع البرمجي أدناه

**Code(10)*

```
$a="84590843058034985093 4";
$b="[0-9]";
if ($a =~ /$b/) {
print "Ok";
}
else {
print "Bad";
}
```

والان عندما يتم تنفيذ هذا المقطع البرمجي الموجود في الاعلى فأن الناتج من عملية التنفيذ سوف تكون كألاتي من خلال التوضيحي أدناه



Figure(7)
Extra Feat

وهذا يعني ان هذه الطريقة من الممكن ان يتم تنفيذها على كافة الارقام مهما كانت كبيرة وذلك لان السطر أدناه

*Code(11)

```
$b="[0-9]";
```

هذا السطر هو عبارة عن سطر شامل لكافة الارقام وبالتالي فأن اي رقم سوف تتم عملية المطابقة معه سوف ينتج عن هذه العملية نجاح من عملية المطابقة

ولكن هذا لايعني ان هذه الخاصية فقط هي خاصية معتمدة على الارقام على العكس حيث من الممكن ان يتم تطبيق هذه الخاصية على الحروف ايضا و المقطع البرمجي ألاتي يوضح الطريقة التي يتم بها التمثيل البرمجي العام لهذه الخاصية ولكن مع الاحرف

*Code(12)

```
$a="iam perl programmer";
$b="[a-z]";
if ($a =~ /$b/) {
print "Ok";
}
else {
print "Bad";
}
```

الان لوتم تنفيذ هذا المقطع البرمجي فأن الناتج من عملية التنفيذ التي سيتم الحصول عليها هي نتيجة تشير بالطبع الى وجود حالة تطابق كما في الشكل التوضيحي أدناه



Figure(8)
Extra Feat Letters

ولكن هذا المقطع البرمجي الموجود في الاعلى من الممكن ملاحظة فقرة هامة وهي ان جميع الاحرف التي تم استخدامها في برمجة هذه الفقرة هي عبرة عن احرف في الحالة الصغيرة لذا عندما يتم ادخال حرف في حالة كبيرة عندها المطابقة لن تسير على ما يرام كما هو موضح في المقطع البرمجي ألاتي

*Code(13)

```
$a="IAM PERL PROGRAMMER";
$b="[a-z]";
if ($a =~ /$b/) {
print "Ok";
}
else {
print "Bad";
}
```

الان عندما يتم تنفيذ هذا الكود فأن الناتج من هذه العملية هي النتيجة التي تشير على عدم وجود مطابقة في البرنامج كما هو موضح في الشكل التوضيحي أدناه

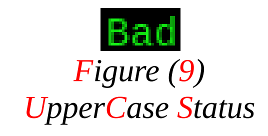


Figure (9)
UpperCase Status

اذن في هذه الحالة اصبح من اللازم على المبرمج ان يكون شامل لهذا الامر من كافة النواحي لذا سوف يتم توسيع المقطع البرمجي أعلاه لكي يكون مقطع اكثر شمولية

*Code(14)

```
$a="IAM PERL PROGRAMMER";
$b="[a-zA-Z]";
if ($a =~ /$b/) {
print "Ok";
}
else {
print "Bad";
}
```

المقطع البرمجي أعلاه اصبحت له القدرة على التعامل مع كافة الاحرف التي تكون بصورة الاحرف في الحالة الكبيرة او الاحرف التي في الحالة الصغيرة واذا تم تنفيذ هذا المقطع البرمجي فأن الناتج من عملية التنفيذ سوف يكون كما يلي من خلال الشكل التوضيحي أدناه



Figure(10)
LetterCase (UC,lc)

How To Ignore

كما ذكر مسبقا فأن البيرل لا تساوي بين حالات الاحرف الكبيرة والصغيرة ولكن لغة البيرل توفر الحل للتخلص من هذه الحالة وذلك باستخدام معرف خاص يقوم بهذه العملية ويكون تمثيله البرمجي كما يلي من خلال المقطع أدناه

*Code(15)

```
$a="perl";
$b="PERL";
if ($a =~ /$b/i){
print "Ok,There is a Match\n";
}
else {
print "No,match\n";
}
```

والان اذا تم تنفيذ هذا المقطع البرمجي فأن الناتج من عملية التنفيذ سوف يكون كما يلي من خلال الشكل التالي أدناه

```
Ok,There is a Match
```

Figure(11)
The i Modifier

اذن يتضح من المقطع البرمجي أعلاه ان المعرف *i* هو المعرف المسؤول عن تجاهل حالة الاحرف وهكذا فأن فأن البيرل تتجاهل حالة الاحرف ولن تفرق بين الاحرف التي تكون في الحالات الكبيرة و الاحرف في الحالات الصغيرة

The s///

هذه الدالة تعتبر من الدوال المبنية في لغة البيرل وتدعى بدالة الاستبدال وغالبا ما يتم استخدامها في التعابير القياسية ويكون التمثيل البرمجي العام لهذه الدالة كما يلي من خلال المقطع البرمجي أدناه

*Code(16)

```
$_="Iam perl programmer";
s/Iam/We are/;
print $_;
```

يوضح المقطع البرمجي أعلاه الكيفية التي من خلالها تم من خلالها استبدال كلمة بكلمة اخرى و الان اذا تم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ سوف تكون كما يلي من خلال الصورة التوضيحية أدناه

```
We are perl programmer
```

Figure(12)
Substitute

The s///g

في المقطع البرمجي السابق تم استخدام دالة الاستبدال بصورة مفردة ولكن هذه الدالة من الممكن ان يتم اضافة بعض الخواص الاضافية عليها لكي تزيد من مرونتها في عمليات الاستبدال ومن المعرفات الهامة التي من الممكن ان يتم اضافتها على دالة الاستبدال هو المعرف *g* و هو يعني *global* ويكون التمثيل البرمجي العام لدالة الاستبدال

هذا المعرف كما يلي من خلال المقطع البرمجي أدناه

*Code(17)

```
$_="Iam perl programmer,And perl my Favorite Lang";
s/perl/P3rL/g;
print $_;
```

الان عندما يتم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ سوف يكون كما يلي من خلال الشكل أدناه

```
Iam P3rL programmer,And P3rL my Favorite Lang
```

Figure(13)
Substitute With g

ان الفائدة من وضع المعرف g في دالة الاستبدال يكون فالغرض هو عندما يتم استبدال كلمة مكان كلمة اخرى فأن هذا الاستبدال يكون على شامل بكلمات اخرى في المقطع البرمجي(16)وردت كلمة perl بهذه الهيئة مرتين وفي دالة الاستبدال تم وضع كلمة P3rLكبديل لكلمة perlالوضع الافتراضي لدالة الاستبدال هو ان تتم عملية الاستبدال مرة واحدة فقط حتى وان كان موجود اكثر من كلمة ولكن عندما تم وضع المعرف g فهذا يعني ان دالة الاستبدال سوف تقوم بتبديل كلمة perl بكلمة P3rLبعدد المرات التي وردت بها كلمة بيرل و هذا يعني ان عملية الاستبدال سوف تتم مرتين فقط لان كلمة perl وردت مرتين فقط ولهذا تمت عملية الاستبدال مرتين

The s/// With Regexp

من الممكن ان يتم استخدام دالة الاستبدال مع تقنية التعبير القياسية لانه التعابير القياسية تقوم على مبدأ اذا تمت المطابقة يتم تنفيذ شرط معين واذا لم تتم المطابقة فشرط اخر سوف يتم تنفيذه وفي هذه الحالة فأن الاسلوب الذي سوف يتم أتباعه هو اذا تمت عملية المطابقة فأن الشرط الذي سوف يتم تطبيقه هو القيام بعملية استبدال و هذه العملية يتم تمثيلها برمجيا كما يلي من خلال المقطع أدناه

*Code(18)

```
$_="perl programming";
if (/perl/){
s/perl/PERL/;
print $_;
}
else {
print "Error";
}
```

اما الان اذا تم تنفيذ هذا المقطع البرمجي فأن الناتج من عملية التنفيذ سوف يكون كما يلي من خلال الصورة الموضحة أدناه

```
PERL programming
```

Figure(14)
Substitute With RegExp

وضح المقطع البرمجي الماضي انه من الممكن ان يتم استعمال دالة الاستبدال مع مفهوم التعابير القياسية وعندما تم حصول شرط المطابقة تم تنفيذ شرط دالة الاستبدال اما اذا لم يتم تحقق الشرط عندها سوف يتم طباعة الجملة التي تشير الى وجود خطأ في عملية المطابقة

Anchors ($)

اغلب البرامج التي تم استخدامها في تقنية التعابير القياسية كان اسلوب تمثيلها البرمجي كما يلي من خلال المقطع البرمجي ألاتي

*Code(19)

```
$a="perl";
$b="programming perl";
if ($a =~ /$b/){
print "OK,There is Match\n";
}
else {
print "Bad Match";
}
```

واذا تم تنفيذ هذا المقطع البرمجي فأن الناتج من عملية التنفيذ هو الناتج الذي يشير الى عدم وجود مطابقة بين المتغير والتعبير القياسي

```
Bad Match
```

Figure(15)
RegExp

يستنتج من المقطع البرمجي أعلاه ان عملية مقارنة كلمة في بداية المتغير الاول مع كلمة في نهاية المتغير الثاني هي عملية غير مقبولة في لغة البيرل بالحالات الاعتيادية ولكن من الممكن ان يتم التخلص من هذه المشكلة باستعمال خصائص اضافية تمكن وتجعل هذه العملية قابلة للتحقيق وهذه الخصائص الاضافية يطلق عليها اسم ال Anchors وهي نوعين
النوع الاول وهو

*Code(20)

```
$
```

هذا هو ال Anchorالاول وتكون الوظيفة المسندة الى هذا ال Anchorهي المقارنة وتكون المقارنة التي يقوم بها هي مقارنة في نهاية التعبير القياسي ويوضح الكود البرمجي ألاتي الطريقة التي يتم تمثيل هذه الخاصية برمجيا

Code(21)

```
$a="perl";
$b="programming perl";
if ($a =~ /$$b/){
print "There is Match\n";
}
else {
print "Bad Match\n";
}
```

الان لو تم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ يكون كما يلي في الشكل التوضيحي أدناه

```
There is Match
```

Figure(16)

Anchor

يوضح المقطع البرمجي السابق نفس الاسلوب العام لتمثيل التعابير القياسية ولكن مع اختلاف واحد فقط و هذا الاختلاف يكمن في الشكل التالي

```
$a="perl";
$b="programming perl";
if ($a =~ /$$b/){
print "There is Match\n";
}
else {
print "Bad Match\n";
```

Figure (17)

Position Anchor $

يلاحظ من الشكل السابق انه يحتوي على

Code(22)

```
$$
```

تعود الاولى الى ال *Anchor*الذي يعمل على المقارنة في نهاية السلسلة النصية

بينما تعود الثانية ال *$b*

Code(23)

```
$a="Freak";
$b="programmingFreak";
if ($a =~ /$$b/){
print "There is Match\n";
}
else {
print "No Match\n";
}
```

الان لو تم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ سوف تكون كما يلي من خلال الشكل التوضيحي أدناه

```
There is Match
```

Figure(18)
Anchor $

يستخلص من الشكل أعلاه ان هذا ال *Anchor* من الممكن ان يقوم بعمليات المطابقة حتى لو كانت الكلمات التي يتم العمل عليها هي كلمات هي منفصلة بفراغات حيث في هذه الحالة يتم عمل مطابقة جزئية بين السلسلة النصية وبين التعبير القياسي

Anchor (^)

هذا النوع الثاني من ال *Anchors*الذي يتم تكون وظيفته هي المطابقة ولكن بصورة تختلف على الطريقة السابقة تكون مهمة هذا ال *Anchor* هي المطابقة من بداية السلسلة النصية ويكون التمثيل البرمجي الخاص لهذا ال *Anchor* هو كما يلي من خلال المقطع البرمجي ألاتي

*Code(24)

```
$a="perlprogramming";
$b="perl";
if ($a =~ /^$b/){
print "Ok.there is match\n";
}
else {
print "No,Match\n";
}
```

اما عن المقطع البرمجي أعلاه فأن الناتج من تنفيذ هذا المقطع هو كألاتي في الشكل التوضيحي أدناه

```
Ok.there is match
```

Figure (19)
Anchor ^

هذا المعرف او هذا ال *Anchor* تكون الوظيفة التي يقوم بها هي المطابقة في بداية السلسلة النصية اي تماما عمله من الناحية البرمجة هو عكس عمل ال *Anchor* السابق

```
$a="perlprogramming";
$b="perl";
if ($a =~ /^$b/){
print "Ok.there is match\n";
}
else {
print "No,Match\n";
}
```

Figure (20)
Position Anchor ^

يوضح من الشكل على انه يحتوي على الرموز

*Code(25)

```
^$
```

حيث ان الرمز الاول يعود الى *Anchor* الذي يعمل على المطابقة في بداية السلسلة النصية
اما الثاني فهو يعود الى التعبير القياسي *$b*

The (.)

في التعابير القياسية المستخدمة في لغة البيرل كانت اسلوب المطابقة بصورة عامة يتم كما يلي من خلال المقطع البرمجي أدناه

*Code(26)

```
$a="Hello World";
$b="Hello";
if ($a =~ /$b/){
print "Match";
}
else {
print "No,Match";
}
```

هذا المطقع البرمجي يقوم على مبدأ مطابقة كلمة بتعبير قياسي واذا تم تنفيذ البرنامج فأن الناتج من عملية التنفيذ لا بد ان يشير الى وجود تطابق في ما بين التعبير القياسي و السلسلة النصية و السبب في هذا ان الكلمة الاولى من السلسة النصية تطابق الكلمة الاولى من التعبير القياسي

اما اذا كان المبرمج يرغب في ان تكون السلسة النصية التي يتعامل معها ان تطابق اي تعبير قياسي فأن لغة البيرل قد وفرت أمكانية تحقيق هذا وذلك من خلال(.)
ويوضح المقطع البرمجي ألاتي الطريقة التي يتم تمثيل هذه التقنية برمجيا

*Code(27)

```
$a="We ArE PErl ProgrammErs";
$b=".";
if ($a =~ /$b/){
print "Match";
}
else {
print "Bad";*Code(27)
}
```

الان اذا تم تنفيذ المطقع البرمجي أعلاه فأن الناتج من عملية التنفيذ سوف يكون كما يلي من خلال الشكل التوضيحي أدناه

```
Match
```

Figure(21)
The .

يلاحظ على المقطع البرمجي أعلاه ان تقنية **الدوت** تعمل على مطابقة السلسلة النصية مهما كانت حالة الاحرف فيها اي سواء كانت الاحرف كبيرة ام صغيرة فأن عملية المطابقة تتم اي ان هذه التقنية تعمل على مطابقة السلسلة النصية مع التعبير القياسي دائما بعض النظر عن ماهية السلسة النصية فأذا تعمل استعمال تقنية الدوت فأن المطابقة سوف تتم الا في حالة واحدة وهذه الحالة هي ممثلة برمجيا كما يلي من خلال المقطع البرمجي ألاتي

*Code(28)

```
$_="\n";
if (/(.)/){
print "Match";
}
else {
print "No,Match";
}
```

اذن تقنية الدوت تكون قادرة على مطابقة كافة السلاسل النصية مع التعبير القياسي الا في حالة كون السلسة النصية هي ال \n ففي هذه الحالة تفشل عملية المقارنة ويكون الناتج من تنفيذ البرنامج هو كما يلي من خلال الشكل التوضيحي أدناه

```
No,Match
```

Figure (22)
Dot Failure

ولكن على الرغم من وجود هذه المشكلة في هذه التقنية ولكن لغة البيرل ايضا قد وفرت الحل لكي يتم التخلص من هذه المشكلة وجعل تقنية الدوت تقنية قادرة على مطابقة كافة السلاسل النصية مع التعبير القياسي النقطي ويتم تمثيل الطريقة التي التخلص منها مشكلة ال *Newline* كما يلي من خلال المطقع البرمجي أدناه

*Code(29)

```
$_="\n";
if (/(.)/s){
print "Match";
}
else {
print "No,Match";
}
```

والن لو تم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ سوف تكون كما يلي من خلال الشكل التوضيحي أدناه

Match

Figure(23)
NewLine problem

The x Modifiers

في لغة البيرل اذا تم تنفيذ مقطع برمجي كالمقطع البرمجي أدناه

*Code(30)

```
$a="perlprogrammer";
$b="perl programmer";
if ($a =~ /$b/){
print "Ok,There is Match";
}
else {
print "No,Match\n";
}
```

الان اذا تم تنفيذ البرنامج السابق فأن الناتج من عملية التنفيذ سوف يكون كما يلي من خلال الشكل التوضيحي أدناه

No,Match

Figure(24)
WhiteSpace

اذن الناتج هو عدم وجود تطابق في البرنامج والسبب في عدم وجود تطابق في البرنامج يكون بسبب

```
$a="perlprogrammer";
$b="perl programmer";
```

Figure(25)
Cause

يوضح الشكل السابق أعلاه سبب الفشل في التطابق لانه في السطر الاول من الشكل السابق يوجد كل من الحرفين *lp* بصورة ملتصقة ولا يوجد اي فراغ بين هذين الحرفين بينما في السطر الثاني يوجد فراغ بين الحرفين *lp* وهذا الفراغ كان السبب في عدم حصول تطابق في البرنامج
ولكن لغة البيرل من خلال التعابير القياسية توفر أمكانية التعامل مع هذه المشكلة وكيفية التخلص منها لذا فعند هذه الحالات يجب على المبرمج ان يفكر بحل بديل وهذا الحل البديل يكمن في المعرف *x*
ويكون التمثيل البرمجي العام لهذا المعرف كما يلي من خلال المقطع البرمجي ألاتي

*Code(31)

```
$a="perlprogrammer";
$b="perl programmer";
if ($a =~ /$b/x){
print "Ok,There is Match";
}
else {
print "No,Match\n";
}
```

الان اذا تم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ تكون كألاتي من خلال الشكل التوضيحي ألاتي

```
Ok,There is Match
```

Figure(26)
x Modifier

اذن وظيفة المعرف *x* هي انه يقوم بعملية الغاء المسافة البيضاء او ما يعرف بال *white space* التي من خلالها الموجودة في التعبير القياسي وعندما تم استعمالها تم الغاء المسافة البيضاء وحصلت المطابقة بين التعبير القياسي و السلسلة النصية

Using Character Class

1-Custom

في لغة البيرل عندما يتم التعامل مع مقطع برمجي يشبه المقطع البرمجي أدناه

*Code(32)

```
$a="cat";
$b="cat";
if ($a =~ /$b/){
print "Ok,There is Match";
}
else {
print "No ,Match";
}
```

فأن الناتج من عملية التنفيذ أدناه سوف تكون بلا شك كما يلي من الشكل التوضيحي المدرج

```
Ok,There is Match
```

Figure (27)
Character Class

لا بد من ان تتم المطابقة في المقطع السابق وذلك بسبب انه كل من التعبير القياسي و السلسلة النصية متطابقين تماما

ولكن اذا كان المبرمج يرغب بأن يقوم بتوسيع رقعة المقارنة التي يقوم بها من خلال لغة البيرل والتعابير القياسية فهذا ممكن من خلال استعمال ال *character class*
ويتم تعريف هذه التقنية على أنها مجموعة من الاحرف او الرموز التي يتم استعمالها لكي يتم الحصول على عدد من التطابقات المحتملة بدلا عن تطابق واحد وعادة يتم وضعه ال *character class* في داخل اقواس مربعة *[]* يكون التمثيل البرمجي لهذه التقنية كما يلي من خلال المقطع البرمجي أدناه

*Code(33)

```
$a="bat";
$b="[crht]at";
if ($a =~ /$b/){
print "Ok,There is Match";
}
else {
print "No ,Match";
}
```

الان اذا تم تنفيذ المقطع البرمجي السابق فأن الناتج من تنفيذه سوف يكون كما يلي من خلال الشكل التوضيحي أدناه

```
Ok,There is Match
```

Figure (28)
Character class using

هذه الطريقة التي يتم استخدام ال *Character class* بها في لغة البيرل واذا القيت نظرة على المطقع البرمجي السابق فيلاحظ ان التعبير القياسي هو كان ال *character class* والسلسلة النصية التي تم استخدامها كانت عبارة عن كلمة *bat* اذن في هذه النقطة يجب ملاحظة أمر ما هو عندما يتم استخدام ال *charcater class* فان الاحرف التي تكون محصورة في داخل الاقواس جميعها تكون قابلة للمطابقة والكلمات التي من الممكن ان يتم استخراجها من المقطع البرمجي أعلاه لكي تكون قابلة للمطابقة ايضا هي

1- bat
2- cat
3- rat
4- hat

Code(34)

```
$a="cat";
$b="[crbt]at";
if ($a =~ /$b/){
print "Ok,There is Match";
}
else {
print "No ,Match";
}
```

وايضا عندما يتم تنفيذ هذا البرنامج فأن ناتج التنفيذ سوف يشير الى وجود تطابق ايضا

```
Ok,There is Match
```

Figure(29)
Character class using

BackSlash Using CharacterClass

كما ذكر سابقا ان ال *character class* يتم استخدامها في التعابير القياسية وذلك من أجل ان يتم توسيع رقعة المطابقة في التعابير القياسية وان تقنية ال *character class* تحتوي على عدد من التقنيات التي تنظم وتحدد عمل هذه التقنية السالفة الذكر المقطع البرمجي أدناه عندما يتم تنفيذه فأن الناتج من عملية تنفيذه سوف تكون الحصول على المطابقة

*Code(35)

```
$a="far";
$b="[cf]ar";
if ($a =~ /$b/){
print "Ok,There is Match";
}
else {
print "No ,Match";
}
```

وناتج تنفيذ هذا المقطع البرمجي أعلاه هو الشكل التالي الذي يشير الى حصول حالة تطابق في البرنامج

```
Ok,There is Match
```

Figure(30)
Character class using

يلاحظ في بعض تقنيات ال *Character class* التي يتم استخدامها في التعابير القياسية وجود الرمز *BackSlash* وأن الغرض من استعمال ال باك سلاش هو حماية المتغير يكون التمثيل البرمجي لهذه التقنية الفرعية كما يلي من خلال المقطع البرمجي أدناه

*Code(36)

```
$a="bcr";
$b="[\$a]at";
if (aat =~ /$b/){
print "There is Match\n";
}
else {
print "No,Match\n"
}
```

عندما يتم تنفيذ المقطع البرمجي ألاتي فأن الناتج من عملية التنفيذ ستكون كما يلي من خلال الشكل التوضيحي أدناه

```
There is Match
```

Figure(31)
BackSlash

ان الغرض من استخدام الباك سلاش هو حماية المتغير اي المقصود بهذه العبارة انه تم اعتماد معيار جديد في عمليات المقارنة لل *Character class* عندما يتم استخدام الباك سلاش والمعيار الذي يتم استخدامه في هذه الحالة هو اشبه بعمليات التوزيع في علم الرياضيات
ان ما يحدث للكود البرمجي عندما يتم استخدام الباك سلاش في البرنامج توجزه الاشكال التوضيحية التالية

```
$a="bcr";تلغى
$b="[\$a]at";
if (aat =~ /$b/){
print "There is Match\n";
}
else {
print "No,Match\n"
}
```

Figure(32)
BackSlash Effect (1)

يوضح الشكل التوضيحي المرقم *32* انه عندما يتم استخدام الرمز باك سلاش فأن الاحرف التي تم اسنادها الى المتغير لن تصبح ذات فائدة بعد الان ولن تعير البيرل اي أهمية لها وذلك لسبب وجود الباك سلاش

```
$b="[\$a]at";
if (aat =~ /$b/){
print "There is Match\n";
}
else {
print "No,Match\n"
}
```

Figure(33)
Matching Possibilities

اذن بعد ان تم الغاء احتمالات التطابق الخاصة بهذا البرنامج بعد ان تم استعمال الباك سلاش كان لابد من ايجاد بديل لها وهذا البديل هو يمكن في الاحتمالات الموضحة بالشكل التوضيحي أعلاه اذ تم اعتماد اسم المتغير ذاته

الذي تم استعماله لكي يكون معيار التطابق الذي يتم العمل على اساسه والمعايير التي تم اعتمادها في المقاطع ال a *character class* السابقة لن يتم اعتمادها في كل كود يحتوي على علامة الباك سلاش بل يتم اعتماد الطريقة التي الموضحة أعلاه

والمعايير التي تحقق التطابق في المقطع السابق هي فقط معيارين هما

1-'$at'

2-aat

تجدر الاشارة الى نقطة هامة الا وهي انه يجب ان يتم حصر الخيار الاول بالعلامة '' وذلك لانه اذا لم يتم حصر هذا الخيار بهذه العلامات فان البيرل سوف تعامل هذا الخيار كما لو انه متغير فعلي لذا يجب استعمال هذه العلامات كي يتم تجنب هذا النوع من الاخطاء

هناك بعض المقاطع البرمجي التي تحتوي على *Double BackSlash* وهذا النوع من المقاطع البرمجية التي تحتوي على *Double BackSlash* فأن الاسلوب المعتمد في المطابقة بها هو الاسلوب القديم ويكون التمثيل لهذه التقنية كما يلي من خلال المقطع ألاتي

*Code(37)

```
$a="bcr";
$b="[\\$a]at";
if ('cat' =~ /$b/){
print "There is Match\n";
}
else {
print "No,Match\n"
}
```

في هذا المقطع البرمجي تم اعتماد الاسلوب القديم في عمليات المطابقة والاحتمالات التطابق في هذا البرنامج هي الاحتمالات ألاتية

1- bat

2- cat

3- rat

واذ تم تنفيذ المقطع البرمجي أعلاه فأن ناتج التنفيذ هو الناتج ألاتي

```
There is Match
```

Figure(34)

Double BackSlash

ومن الممكن ان يقوم المبرمج بتوسيع رقعة المقارنة التي يعمل عليها في تقنية ال *Character class* ويتم ذلك عن طريق استخدام ال *Anchors*

*Code(38)

```
$
```

*Code(39)

```
^
```

بنفس الطريقة التي تم التطرق اليها سابقا

Using Character Class

2-classic

يتضمن الشق الثاني من تقنية *Character Class* العديد من المعرفات التي التي من شأنها ان تعمل على تسهيل ال
عمليات المقارنة التي تتم من قبل البرمج وهذا النوع من ال *Character Class* يحتوي على عدد من المعرفات والمعرف الاول هو

1-\w

المعرف البرمجي أعلاه يقوم بوظيفة هامة من شأنها ان تسهل بالكثير من عمليات المقارنة التي تتم في تقنية التعابير القياسية هذا المعرف يقوم بمطابقة جميع السلاسل النصية المعطاة له مهما كان نوع هذه السلاسل النصية يكون التمثيل البرمجي لهذا المعرف كما يلي من خلال المقطع البرمجي أدناه

*Code(40)

```
$_="We Are Perl Programmer";
if (/\w/){
print "There Is Match";
}
else {
print "No match Found";
}
```

الان اذا تم تنفيذ البرنامج فأن الناتج من عملية تنفيذ البرنامج سوف تكون كما يلي من خلال الشكل التوضيحي أدناه

```
There Is Match
```

Figure(35)
\w

وضح المقطع البرمجي أعلاه أنه المعرف *\w* هو معرف مسؤول دأئما عن مطابقة السلاسل النصية مهما كانت حالة الاحرف المستخدمة في البرنامج فهذا المعرف يعمل على مطابقة الاحرف الكبيرة و الصغيرة فهي سيان لهذا المعرف
وهذا المعرف هو مختصر لكلمة *word* ولهذا فهو يطابق جميع الاحرف والكلمات التي يتم استخدامها في البرنامج

2- \d

هذا المعرف الثاني من المعرفات التي تحتويها تقنية ال *Character Class* وهذا المعرف يقوم بوظيفة مطابقة الارقام حيث انه يقوم بمطابقة الارقام المعطاة في البرنامج مهما كانت فهو كفيل بتحقيق المطابقة ويكون التمثيل البرمجي لهذا المعرف كما يلي من خلال المقطع البرمجي أدناه

Code(41)

```
$_="123456789";
if (/\d/){
print "There Is Match";
}
else {
print "No match Found";
}
```

الان اذا تم تنفيذ البرنامج فأن الناتج من تنفيذ البرنامج سوف يكون كما يلي من خلال الشكل التوضيحي أدناه

```
There Is Match
```

Figure(36)
\d

وضح المقطع البرمجي أعلاه ان المعرف الذي يحمل الاسم *\d* يقوم بمطابقة الارقام التي تعطى له ويتعامل مع هذه الارقام وعندها يتم الحصول على التطابق من البرنامج
تجدر الاشارة الى أن اسم هذا المعرف مشتق من كلمة *Digit* والتي تعني رقم أو عدد رياضي

3- \s

المعرف الاخير ضمن هذه الفئة من تقنية ال *Character Class* يقوم بوظيفة لا تقارن بأهمية الوظائف التي تقوم بها المعرفات السابقة ويكون التمثيل البرمجي الخاص بهذا المعرف كما يلي من خلال المقطع البرمجي أدناه

**Code(42)*

```
$_=" \t     \n";
if (/\s/){
print "There Is Match";
}
else {
print "No match Found";
}
```

ألان أذا تم تنفيذ المقطع البرمجي أدناه فأن الناتج من عملية التنفيذ سوف تكون كما يلي من خلال الشكل التوضيحي أدناه

```
There Is Match
```

Figure(37)

\s

الوظيفة البرمجية التي يقوم بها هذا المعرف هي أنه يقوم بمطابقة الفراغات والمسافات البيضاء التي تعطى له في المقطع البرمجي تجدر الاشارة الى ان المعرف *\s*هو أختصارا لكلمة *Whitespace*والتي تعني فراغ او مسافة بيضاء

Character Class

3- Negation

يتضمن الشق الثالث و الاخير من هذه التقنية مبدأ او اسلوب الانكار الذي يعمل تماما بأسلوب معاكس للتقنية السابقة و هذا الشق يتضمن ايضا 3 أنواع من المعرفات التي تعمل كما يلي

1- \W

أن المعرف الذي كان يحمل الاسم \w كان له وظيفة محددة و هي أن يعمل على مطابقة الاحرف و السلاسل النصية التي تعطى له في البرنامج ويقوم بمطابقتها اما هذا المعرف فهو تماما يعمل على عكس ما كان يفعل المعرف \w حيث أن هذا المعرف لا يعمل على مطابقة الاحرف و السلاسل النصية يكون التمثيل البرمجي الخاص به بنفس اسلوب وطريقة التمثيل البرمجي الخاص بالمعرفات السابقة

*Code(43)

```
$_="Perlprogramming";
if (/\W/){
print "There Is Match";
}
else {
print "No match Found";
}
```

وناتج تنفيذ المقطع البرمجي أعلاه هي تكون كما يلي من خلال الشكل التوضيحي ألاتي

```
No match Found
```

Figure(38)

\W

يلاحظ على هذا المعرف انه لا يمتلك القدرة على التعامل مع الحروف و السلاسل النصية لذا ان صادف وان تعامل معها فأن النتيجة التي سوف يحصل عليها المبرمج هي عدم حصول التطابق ولكن له القدرة على التعامل مع الفراغات والمساحات البيضاء

*Code(44)

```
$_="        ";
if (/\W/){
print "There Is Match";
}
else {
print "No match Found";
}
```

يوضح المقطع البرمجي الموجود في الاعلى ان لهذا المعرف القدرة على التعامل مع الفراغات و المساحات البيضاء

```
There is Match
```

Figure(39)

\W

2- \D

المعرف الذي يحمل الاسم \d كانت له وظيفة محددة وهذه الوظيفة التي يقوم بها هي مقارنة الارقام حيث ان الارقام التي يتم مطابقتها مع هذا المعرف يتم الحصول مباشرة على التطابق أما المعرف أعلاه \D فهو يعمل عكس المعرف \d الذي اذا تمت مطابقته مع الارقام فأن الناتج من عملية التطابق سوف لن يشير الى حدوث تطابق ويكون التمثيل البرمجي الخاص بهذا المعرف كما يلي من خلال المقطع البرمجي ألاتي

*Code(45)

```
$_="123456789";
if (/\D/){
print "There is Match";
}
else {
print "No Match Found";
}
```

الان اذا تم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ سوف تكون كما يلي من خلال الشكل التوضيحي ألاتي

```
No Match Found
```

Figure(40)
\D

ولكن هذا المعرف اذا تمت معاملته مع سلسلة نصية او حروف فأن التمثيل البرمجي لهذه العملية تكون كما يلي من خلال المقطع البرمجي أدناه

*Code(46)

```
$_="Perl Programming";
if (/\D/){
print "There is Match";
}
else {
print "No Match Found";
}
```

ناتج تنفيذ المقطع البرمجي أعلاه يكون كما يلي من خل الشكل التالي الذي يشير ألى حصول التطابق في البرنامج وعلى قدرة تعامل هذا المعرف مع المعطيات ما لم تكون معطيات رقمية او أعداد

```
There is Match
```

Figure(41)
\D

3- \S

هذا المعرف هو المعرف الاخير في هذه الفئة و المعرف الاخير في عموم تقنية ال *Character Class* وهذا المعرف هو كباقي المعرفات السابقة الذكر حيث أن هذا المعرف لايملك القدرة لكي يتعامل مع الفراغات و المساحات البيضاء واذا تمت معاملته بها فأن نتيجة تنفيذ البرنامج تشير الى عدم حصول التطابق ويكون تمثيله البرمجي هو كألاتي من خلال المقطع البرمجي ألاتي

**Code(47)*

```
$_="\t\n  ";
if (/\S/){
print "There is Match";
}
else {
print "No Match Found";
}
```

وناتج تنفيذ هذا البرنامج تكون هي كألاتي من خلال الشكل التوضيحي ألاتي

```
No Match Found
```

Figure(42)
\S

ومن الممكن ان يتم ومعاملة هذا المعرف بالسلاسل النصية والمعرفات الرقمية كما يلي من خلال المقطع البرمجي ألاتي

**Code(48)*

```
$_="Perl Programming";
if (/\S/){
print "There is Match";
}
else {
print "No Match Found";
}
```

وعندما يتم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ تشير بصورة قطعية الى حصول التطابق لان هذا المعرف له القدرة البرمجية على التعامل مع هذه المعطيات

```
There is Match
```

Figure(43)
\S

يوجز الجدول ألاتي جميع المعرفات التي تحتويها تقنية ال *Character Class*

المعرف	الوظيفة	المختصر	التعارض
\w	هذا المعرف يقوم بمطابقة جميع الحروف و السلاسل النصية	word	لايتوافق مع white spaceأو الفراغات البيضاء ال
\d	هذا المعرف يقوم بمطابقة جميع المعطيات اذا كانت عبارة عن ارقام رياضية	digit	هذا المعرف لا يتوافق مع السلاسل النصية و الحروف وكما ليست له القدرة على التعامل مع الفراغات و ال white spaces
\s	هذا المعرف الوظيفة البرمجية المسندة اليه هي مطابقة الفراغات والمساحات البيضاء اي ما يعرف بال White space	White space	لا يتم الحصول على حالة تطابق اذا تمت مطابقة هذا المعرف مع السلاسل النصية او مع الارقام
\W	يمتلك هذا المعرف القدرة على التعامع مع الفراغات و المساحات البيضاء	NonWord	لايمتلك القدرة والامكانية على التعامل مع السلاسل النصية و الحروف
\D	يمتلك هذا المعرف القدرة على التعامل مع المعرفات الحرفية او السلاسل النصية وكذلك يتملك هذا المعرف القدرة على التعامل مع المساحات الفارغة و الفراغات	NonDigit	لايمتلك هذا المعرف القدرة على التعامل مع المعطيات الرقمية او الارقام التي تعطى له في البرنامج
\S	يمتلك هذا المعرف القدرة على التعامل مع الارقام و السلاسل النصية	NonWhite Space	لايمتلك هذا المعرف القدرة على التعامل مع المساحات البيضاء و الفراغات في البرنامج

Positions

سبق وان تم التطرق الى ال *Anchors*الخاصة بعمليات التطابق وكانت هذه ال *Anchors* هي

*Code(49)

```
^
```

هذا ال Anchor مسؤول عن عمليات المطابقة في بداية السلسلة النصية

*Code(50)

```
$
```

أما عن هذا ال *Anchor* فهو يكون مسؤول عن عمليات التطابق في نهاية السلاسل النصية اي كل من هذين ال *Anchors* يعمل أحدهما عكس الاخر

ولكن على الرغم من ان لغة البيرل وفرت هذه ال *Anchors* ولكن اضافة لها قد وفرت ايضا عدد من المعرفات التي تقوم بنفس المهمة التي تقوم بها ال *Anchors* و هذه المعرفات هي

*Code(51)

```
\A
```

هذا المعرف هو مشابه لل *Anchor*الذي يحمل السمة (^)

ويكون التمثيل البرمجي لهذا المعرف كما يلي من خلال المقطع البرمجي أدناه

*Code(52)

```
$_="perlprogramming";
if (/\Aperl/){
print "There is Match";
}
else {
print "No match Found";
}
```

الان أذا تم تنفيذ المقطع البرمجي أعلاه فأن النتج من عملية التنفيذ ستكون كما يلي من خلال الشكل التوضيحي أدناه

```
There is Match
```

Figure(44)

\A

يوضح المقطع البرمجي أعلاه أن المعرف الذي يحمل الاسم *\A* هو عبارة عن معرف يقوم بعمليات المقارنة التي الخاصة ببداية السلسلة النصية و هو عادة قليل الاستخدام وذلك لأن اغلب مبرمجي البيرل يميلون الى استخدام ال *Anchors* بدلا عن هذه المعرفات وايضا لان ال *Anchors*هي اكثر سهولة وبساطة في الاستخدام

The \z

المعرف z\ هو عبارة عن معرف يقوم بعمليات المطابقة في نهاية السلاسل النصية ويكون التمثيل البرمجي لهذا المعرف كما يلي من خلال المقطع البرمجي أدناه

*Code(53)

```
$_="perlprogramming";
if (/programming\z/){
print "There is Match";
}
else {
print "No match Found";
}
```

الان عندما يتم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ سوف تكون كما يلي من خلال الشكل التوضيحي ألاتي

```
There is Match
```

Figure(45)
\z

The \b and \B

1- \b

هذين المعرفين هم مختصر لكلمة *boundary*والتي تعني حدود ويكون المعرف *b* أعلاه يعمل على مبدأ مطابقة كألاتي *w*والتي تمثل الحرف او الرمز و *W* والتي يتم في هذه الحالة يتم أعتبارها نهاية الجملة او نهاية السلسلة النصية ويكون التمثيل البرمجي العام لهذا المعرف هو كما يلي من خلال المقطع البرمجي أدناه

*Code(54)

```
$_="perlprogramming";
if (/programming\b/){
print "There is Match";
}
else {
print "No match Found";
}
```

الان عندما يتمتنفيذ هذا البرنامج فأن الناتج من عملية التنفيذ سوف تشير الى حدوث التطابق كما يلي من خلال الشكل الوضيحي أدناه

```
There is Match
```

Figure(46)

\b

بألقاء نظرة على المقطع البرمجي أعلاه عندها من الممكن ان يتم ملاحظة ما يلي

```
$_="perlprogramming";
if (/programming\b/){
print "There is Match";
}
else {
print "No match Found";
}
```

Figure(47)

\b

يوجز الشكل أعلاه ان السلسلة المظللة بالمربع الاحمر كان لابد ان تكون في نهاية الجملة او في نهاية السلسلة النصية لكي تتم عملية المطابقة بشكل ناجح لانه لو كانت الكلمة المظللة بالمربع الاحمر بهذه الصورة لكانت عملية المطابقة لن تحدث ولكان هنالك خطأ برمجي في المقطع البرمجي

```
$_="perlprogrammingisfun";
if (/programming\b/){
print "There is Match";
}
else {
print "No match Found";
}
```

Figure(48)

\b

وضح الشكل التوضيحي أعلاه كلمة *programming* هي ليست نهاية السلسلة النصية بل يوجد بعدها تكلمة وهذه التكملة هي حروف وعند هذه المرحلة عندما يقوم المبرمج بتنفيذ البرنامج فأنه سوف يحصل على النتيجة ألاتية

```
No match Found
```

Figure(49)

\b

ولكن اذا كانت هذه الاضافة ضرورية ولايمكن الاستغناء عنها فأنه من الممكن ان يتم اضافية تعديل برمجي بسيط لن يضر بالبنية البرمجية الخاصة بالنطام وهذا التعديل يكون كألاتي

**Code(55)*

```
$_="perlprogramming isfun";
if (/programming\b/){
print "There is Match";
}
else {
print "No match Found";
}
```

الان عندما يتم تنفيذ البرنامج أعلاه فان الناتج من عملية التنفيذ هي النتيجة ألاتية التي تشير الى حصول تطابق في البرنامج

```
There is Match
```

Figure(50)

\b

على الرغم من ان التعديل الذي تمت اضافته الى البرنامج هو عبارة عن تعديل بسيط يكمن في أضافة فراغ واحد الى البرنامج ولكن هذا الفراغ الذي تمت اضافته الى البرنامج أعلاه فأنه في تقنية ال Character Class

فأن الفراغ يتم تصنيفه على أنه \s

وذكر في الاعلى ان هذا المعرف يعمل على مبدأ او قانون

*Code(56)

```
\w ...... \W
```

2- \B

هذا المعرف ايضا يعمل على وفق نفس المبدأ ولكن مع بعض الاختلافات الطفيفة من ناحية العمل يكون المبدأ العام او القانون الذي يسير عليه هذا المعرف هو القانون ألاتي

*Code(57)

```
\w ...... \w
```

ويكون التمثيل البرمجي العام لهذا المعرف هو كما يلي من خلال المقطع البرمجي ألاتي

*Code(58)

```
$_="perlprogrammingisfun";
if (/\Bis\B/){
print "There is Match";
}
else {
print "No match Found";
}
```

والان اذا تم تنفيذ البرنامج فان ناتج تنفيذه سوف يكون كما يلي من خلال الشكل التوضيحي أدناه

```
There is Match
```

Figure(51)

\B

Capturing

التعابير القياسية تسمح للمبرمج بوضع أجزاء من من ال *patterns* في *patterns* فرعية ويتم خزنها في متغيرات لكي يتم استدعائها في وقت اخر عندما تدعو الحاجة لها و هذه التقنية يطلق عليها *Capturing*

Capturing

هي عملية التقاط جزء من السلسلة النصية ويتم وضعها في متغير أخر لاستعمال لاحق ويتم استعمال الاقواس () في التعريف العام لهذه التقنية ويكون التمثيل البرمجي العام لهذه التقنية هو كألاتي

**Code(59)*

```
$_="PerlProgramming";
if (/(\w)(\w)(\w)(\w)/){
print "The First  Letter is:-$1","\n";
print "The Second Letter is:-$2","\n";
print "The Third  Letter is:-$3","\n";
print "The Fourth Letter is:-$4","\n";
}
```

الان عندما يتم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية تنفيذه ستكون كما يلي من خلال الشكل التوضيحي ألاتي

```
The First  Letter is:-P
The Second Letter is:-e
The Third  Letter is:-r
The Fourth Letter is:-l
```

Figure(52)

Capture

في المقطع البرمجي أعلاه تم استخدام تقنية *Character Class*وذلك من اجل ان يتم التقاط حرف من السلسلة النصية المستخدمة في المقطع البرمجي حيث ان كل قوس من*(\w)* يعمل على التقاط حرف وباستخدام اربع اقواس فأن النتيجة هي التقاط أربع أحرف من السلسلة وكانت النتيجة هي ظهور كلمة *perl* عند تنفيذ البرنامج وفي كل قوس تم استخدامه يقوم هذا القوس الاول بتحويل الحرف الذي يحتويه الى المتغير *$1*

ثم يقوم القوس الثاني بتحويل الحرف الذي يحتويه الى المتغير الثاني *$2*

ثم يقوم القوس الثالث بتحويل الحرف الذي يحتويه الى المتغير الثالث *$3*

ثم يقوم القوس الرابع بتحويل الحرف الذي يحتويه الى المتغير الرابع *$4*

يحتوي المتغير الاول *($1)* على الحرف الاول وهو الحرف *p*

يحتوي المتغير الثاني ($2) على الحرف الثاني وهو الحرف e

يحتوي المتغير الثالث ($3) على الحرف الثالث وهو الحرف r

ويحتوي الحرف الرابع ($4) على الحرف الرابع وهو الحرف l

ومن الممكن ان يتم تطبيق هذه التقنية ليس على الحرف فقط ولكن من الممكن ان يتم تطبيقها على الارقام ويكون الاسلوب البرمجي الذي يتم به هذا التمثيل هو كما يلي

*Code(60)

```
$_="123456789";
if (/(\d)(\d)(\d)(\d)/){
print "The First  Digit is:-$1","\n";
print "The Second Digit is:-$2","\n";
print "The Third  Digit is:-$3","\n";
print "The Fourth Letter is:-$4","\n";
}
```

ويكون ناتج تنفيذ المقطع البرمجي أعلاه كما يلي من خلال الشكل التوضيحي أدناه

```
The First  Digit is:-1
The Second Digit is:-2
The Third  Digit is:-3
The Fourth Letter is:-4
```

Figure(53)

Capturing

أما عن الطريقة البرمجية التي تم تنفيذ الكود بها فهي الطريقة عينها التي تم ذكرها في عندما تم شرح المقطع البرمجي المرقم بالرقم *Code(58)

يلاحظ في بعض المقاطع البرمجية التي تستعمل تقنية ال capturing أنها تحتوي على علامة(+)

ومن هذه المقاطع التي تحتوي على الكود البرمجي التالي

*Code(61)

```
$_="123456789";
if (/(\d+)/){
print "The All Digits Are:-$1","\n";
}
```

عندما يتم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ تكون كما يلي من خلال الشكل التوضيحي ألاتي

```
The All Digits Are:-123456789
```

Figure(54)

Capturing With +

لو كان المقطع البرمجي أعلاه لا يحتوي على علامة الجمع (+) لكان الناتج من عملية التنفيذ سوف يكون هو طباعة الرقم الاول من السلسلة النصية و هذا الرقم سيكون الرقم 1

ولكن نظرا لوجود العلامة الخاصة بالجمع فأن هذه العلامة تقوم بأخذ كافة العناصر الموجودة في السلسلة النصية المستخدمة مهما كان طول هذه السلسلة وارسالها الى المتغير $1 الذي سوف يستلم البيانات

ومن الممكن ان يتم استعمال هذه التقنية أي تقنية ال+ Capturing With مع الحروف و السلاسل النصية والمقطع البرمجي ألاتي يوضح الطريقة التي يتم اتباعها مع السلاسل النصية

*Code(62)

```
$_="Perl programming is Fun";
if (/(\w+)(\w)(\w)(\w)/){
print $1,"\n";
print $2,"\n";
print $3,"\n";
print $4,"\n";
}
```

الان عندما يتم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ تكون كما يلي من خلال الشكل التوضيحي ألاتي



Figure(55)

Capturing With +

ولكن النتيجة التي تم الحصول عليها من خلال هذا البرنامج لم تكن هي النتيجة المتوقعة كان من المفترض أن يكون الناتج لتنفيذ المقطع البرمجي السابق هو الناتج ألاتي



Figure(56)

Capturing

السبب في الحصول على النتيجة الموضحة بالشكل 55 كان سبب بسيطا وهو وجود فراغ بين مقاطع السلسلة النصية المستخدمة في البرنامج

```
$_="Perl Programming Is Fun";
```

Figure(57)

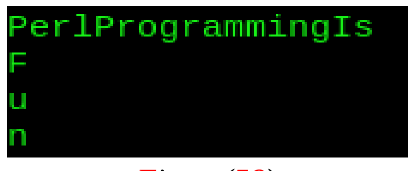
Dash

السبب هو وجود فراغ بين كلمة *perl* وكلمة *programming* ويلاحظ في الشرط الخاص بالبرنامج ان ال *Character Class* الذي تم أستخدامه في البرنامج هو *w*و الفراغ الذي في البرنامج لا يعود الى فئة w\ ولكن يعود الى فئة s\ لانه فراغ
ولكي يحصل المبرمج على نتيجة مشابهة للنتيجة الموضحة بالشكل*Figure(56)* فهذا يتم عن طريق أجراء تعديل برمجي بسيط كألاتي في المقطع البرمجي أدناه

*Code(63)

```
$_="PerlProgrammingIsFun";
if (/(\w+)(\w)(\w)(\w)/){
print $1,"\n";
print $2,"\n";
print $3,"\n";
print $4,"\n";
}
```

الان عندما يتم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ سوف يكون كما يلي من خلال الشكل التوضيحي ألاتي



Figure(58)

Capturing With +

يوضح المقطع البرمجي أعلاه ان القوس البرمجي الذي يحمل السمة *(\w+)* يقوم بعمل التقاط لكافة الحروف المكونة للسلسلة النصية ولكن يلاحظ ان هذا القوس يتوقف عندما يتم التوصل الى أخر *3* أحرف من السلسلة النصية والسبب وراء ذلك هو وجود *3* اقواس من بعد القوس*(\w+)* والتي تعمل على التقاط الحروف الثلاثة الاخيرة من السلسلة

الحالة الاخيرة التي يتم التطريق اليها هي الحالة التي يكون القوس البرمجي بحالة مشابهة لهذه الحالة

*Code(64)

```
(\w)+
```

تحتوي بعض المقاطع البرمجية التي تستعمل تقنية ال *Capturing*أن القوس البرمجي الذي المستخدم في البرنامج

يكون مشابه للقوس الموضح في المقطع البرمجي*Code(63)* وتمثل هذه التقنية برمجيا كما يلي من خلال المقطع البرمجي ألاتي

*Code(65)

```
$_="PerlProgrammingIsFun";
if (/(\w)+/){
print $1,"\n";
}
```

ألان عندما يتم تنفيذ المقطع البرمجي أعلاه فأن النتيجة التي سيتم الحصول عليها هي كألاتي من خلال الشكل التوضيحي أدناه



Figure(59)

Capturing (\w)+

النتيجة البرمجية التي يتم الحصول عليها من المقطع البرمجي أعلاه هي الحصول على الحرف الاخير الموجود في السلسلة النصية

```
$_="PerlProgrammingIsFun";
```

Figure(60)

Capturing (\w)+

Alternation

يتم استخدام هذه التقنية في بعض الحالات التي يكون المطلوب منها تكوين عدد من الاحتمالات في عملية المطابقة ويستخدم الرمز **|** للتعبير او للتعريف عن هذه التقنية البرمجية ويكون التمثيل البرمجي الخاص بها هو كألاتي في المقطع البرمجي أدناه

*Code(66)

```
$_="fill";
if (/f|k|w|h|ill/){
print "There Is match";
}
else {
print "No Match";
}
```

الان عندما يتم تنفيذ البرنامج يتم الحصول على نتيجة مشابهة للنتيجة الموضحة في الشكل التالي

```
There Is match
```

Figure(61)

Alternation

وفي المقطع البرمجي أعلاه فأن قائمة الاحتمالات التي عندما يتم استخدامها يتم الحصول على التطابق هي

1- fill

2- kill

3- will

4- hill

في الحالات التي تكون السلسلة النصية مطابقة للتعبير القياسي الاول والتعبير القياسي الثاني ففي هذه الحالة تكون الاولوية للخيار الاول ومن بعد ذلك يأتي الخيار الثاني وهكذا من الممكن ان يتم تمثيل هذه الحالة البرمجية كما يلي من خلال المقطع البرمجي ألاتي

*Code(67)

```
$_="perlprog";
if (/perl|programming|programmer/){
print "There Is match";
}
else {
print "No Match";
}
```

عندما يتم تنفيذ المقطع البرمجي أعلاه فأن الناتج من عملية التنفيذ هي تشير الى التطابق وكما يلي من خلال الشكل التوضيحي ألاتي

```
There Is match
```

Figure(62)

Possibilities

يوضح المقطع البرمجي أعلاه ان البرنامج يحتوي على السلسلة النصية

**Code(68)*

```
perlprog
```

والتعبير القياسي الذي يحتويه البرنامج هو التعبير القياسي ألاتي

**Code(69)*

```
/perl|programming|programmer/
```

يستخلص من المقطع البرمجي *Code(67)* والمقطع البرمجي *Code(68)* ان التطابق بين السلسلة النصية

والتعبير القياسي هو في أول *8* أحرف

```
$_='perlprog';
if (/perl|programming|programmer/){
```

Figure(63)

Matching

يوضح من الشكل التوضيحي ان نسبة التطابق هي فقط في اول ثمانية أحرف ولكن عندما يقوم البرنامج ولكن عندما يقوم محرك بحث البيرل بالبحث عن التطابق فأنه في هذه الحالة سوف يكون الخيار الاول على الرغم من ان الخيار الثاني من التعبير القياسيي يطابق ايضا اول *8* احرف من السلسلة النصية ولكن البيرل تعمد على اختيار التطابق الذي تحصل عليه أولا وفي هذه الحالة سوف يكون *perl programming* وعلى سبيل الافتراض اذا كان

الخيار الاول لا يسبب التطابق سوف تتوجه البيرل الى الخيار الثاني وهكذا الى ان يتم الحصول على التطابق

من الممكن أن يتم اضافة *Extra feature* الى تقنية ال *Alternation* مما يزيد من الامكانيات

المتاحة لاستعمال هذه التقنية ومن الميزات التي يتم اضافتها الى هذه التقنية هي ميزة ال( ) والمقطع البرمجي ألاتي يوضح الكيفية التي يتم تمثيل هذه الميزة برمجيا

**Code(70)*

```
$_="perlgeek";
if (/perl(programmer|geek)?/){
print "There Is match";
}
else {
print "No Match";
}
```

الان عندما يتم تنفيذ المقطع البرمجي أعلاه فان الناتج من عملية تنفيذ البرنامج اعلاه سوف تكون كما يلي من

خلال الشكل التوضيحي ألاتي

There Is match

Figure(64)

? Feature

وفي الكود أعلاه فأن قائمة الاحتمالات التي عندما يتم استخدامها يتم الحصول على التطابق هي

1- perl

2- perlprogrammer

3- perlgeek

ولكن مما تجدر الاشارة اليه ان المقطع *Code(69)** يحتوي على ميزة اخرى في محتوياته هي ال *capturing*

```
if ((/perl(programmer|geek)
```

Figure(65)

capturing with ?

اذن عندما يحتوي البرنامج على تقنية ال *capturing* فهذا يعني أن المبرمج له القدرة على معرفة ما هو التطابق الذي تم الحصول عليه من داخل الاقواس البرمجية التي يحتويها البرنامج اذن الان من الممكن ان يتم اعادة كتابة البرنامج السابق مع أمكانية معرفة ماهو التطابق الذي سيتم الحصول عليه من داخل الاقواس الخاصة بتقنية التقاط

**Code(71)*

```
$_="perlgeek";
if (/perl(programmer|geek)?/){
print "There Is match\n";
print $1,"\n";
}
else {
print "No Match";
}
```

والان عندما يتم تنفيذ البرنامج فان الناتج من عملية التنفيذ سيكون الحصول على التطابق و الحصول على الخيار الذي تمت مطابقته من داخل الاقواس التي يحتويها البرنامج



Figure(66)

Execution Result

اذن الوظيفة التي يقوم بها المعرف ؟ هي السماح للكلمة المفتاحية المستخدمة في التعبير القياسي ان تكون جزء قابل للمقارنة أي كما يلي من خلال المقطع البرمجي الاتي

*Code(72)

```
$_="perl";
if (/perl(programmer|geek)?/){
print "There Is match\n";
print $1,"\n";
}
else {
print "No Match";
}
```

الان عندما يتم تنفيذ البرنامج يلاحظ ان النتيجة التي يتم الحصول عليها من عملية التنفيذ هي النتيجة الاتية و التي تشير الى حدوث تطابق في البرنامج

```
There is Match
```

Figure(67)

KeyWord

Backtrack

ان عملية البحث عن التطابق والانتقال الى الحروف او الكلمة الاخرى ان لم يتم الحصول على التطابق يطلق على هذه العملية عملية ال *backtrack* يتم تمثيل هذه العملية برمجيا كما يلي من خلال المقطع البرمجي الاتي

**Code(73)*

```
$_="abcde";
if (/(abd|abc)(df|d|de)/){
print "There Is Match";
}
else {
print "There`s no match";
}
```

عندما يتم تنفيذ البرنامج اعلاه فان الناتج من التنفيذ سوف يشير الى حدوث التطابق ولكن المهم في هذا المقطع هو الكيفية التي تتم بها عمليات التطابق في التعابير القياسية وسيتم تلخيص عملية المقارنة بالتعابير القياسية بالنقاط الاتية

أولا : يتم البدء بالحرف الاول من السلسلة النصية في التعبير القياسي

ثانيا : يتم أخذ المجموعة الاولى من ال *Alternation list* وهذه المجموعة هي *abd*

ثالثا : الحرف الاول من السلسلة النصية يطابق الحرف الاول من مجموعة ال ***a**bd* والحرف الثاني من السلسلة النصية يطابق الحرف الثانية من مجموعة ال *a**b**d*

رابعا : الان الحرف الثالث الموجود في السلسلة النصية وهو الحرف *c* لا يطابق الحرف الثالث من التعبير القياسي اذن في هذه الحالة يوجد *DeadEnd* لذا سيتم عمل *BackTrack* بحرفين تم الحصول على التطابق لهم وهذين الحرفين هما ***ab***

خامسا : سيتم الانتقال الى المجموعة الثانية من ال *Alternation list* وهذه المجموعة هي *abc*

سادسا : سيتم الحصول على التطابق من المجموعة الثانية وهذا يعني ان الحرف c سوف يتم مطابقته مع الحرف الثالث من المجموعة الثانية *Alternation list*

سابعا : يتم الانتقال الى المجموعة الثانية وهي مجموعة ال *df* عندها سيتم الحصول على تطابق بين التعبير القياسي و الحرف *d* في السلسلة النصية

ثامنا : الحرف *f* في التعبير القياسي لا يطابق الحرف *e* في السلسلة النصية لذا في هذه الحالة أيضا يوجد *DeadEnd* وسيحصل *BackTrack* والانتقال الى المجموعة الاخرى وهي حرف ال *d* وبما انه حرف ال *d* قد حصل على التطابق الخاص به لذا سوف يتم الانتقال الى المجموعة الاخرى و الاخيرة من التعبير القياسي وهي مجموعة *de*

تاسعا : الحرف الثاني من المجموعة الاخيرة من التعبير القياسي يطابق الحرف الاخير من السلسلة النصية وعندها سيتم الحصول على التطابق الكامل بين التعبير القياسي و السلسلة النصية المعطاة في المقطع البرمجي

The Last Sp3ll

*License

This Book is Created under the terms of the GPL (General public License)

*Greets

Greets for (Binary,Striker,Storm,SNIX,Raidy,Enc,SoFy.Mutati0n)

*Special Thanks

To the person who always support me and always put me on the right way and be there for me whenever i want god bless you and thanks for your great support

But here I am, on the road again
here I am, up on the stage
here I go, playing the star again
there I go, turn the page

Wrote By:-

M_SpAwN

(Black Saint)